JTRS Input/Output API Service Definition

V1.0
December 15, 2000

Prepared for the
Joint Tactical Radio System (JTRS) Joint Program Office

Prepared by the
Modular Software-programmable Radio Consortium
Under Contract No. DAAB15-00-3-0001

Revision Summary

1.0	Initial release

Table Of Contents

List of Figures

List of Tables

1 INTRODUCTION.

1.1 OVERVIEW.

This document specifies an SCA conformant IO layer Application-Programming Interface (API) service definition. It provides the interface definition for Audio and Data I/O Service providers. This API provides the following Non-Real-Time Services:

1. I/O Configuration: provides initial or subsequent configuration of a data device.
2. I/O Control: provides operational control of Audio and Data Devices.
3. Audible Alerts and Alarms: provides a Service User the ability to define and control audible alerts and alarms generated by an I/O Audio device.
4. I/O Signals: allows an I/O device to signal a request to transmit (PTT) to other APIs,

and the following Real-Time Services:

5. Packet: provides the mechanism to establish packet queues and exchange data packets with I/O Devices.
6. I/O Error Signal: provides a mechanism for a packet Service Provider to inform a Service User when a packet contains errors.

The I/O API is constructed by instantiating the SCA Audio I/O Building Block and SCA Generic Packet Building Block with the specific parameter types.

Waveform Service-layers are shown in Figure 1-1. Services defined in this API are used in conjunction with APIs for other layers to form a complete application-programming interface (API).

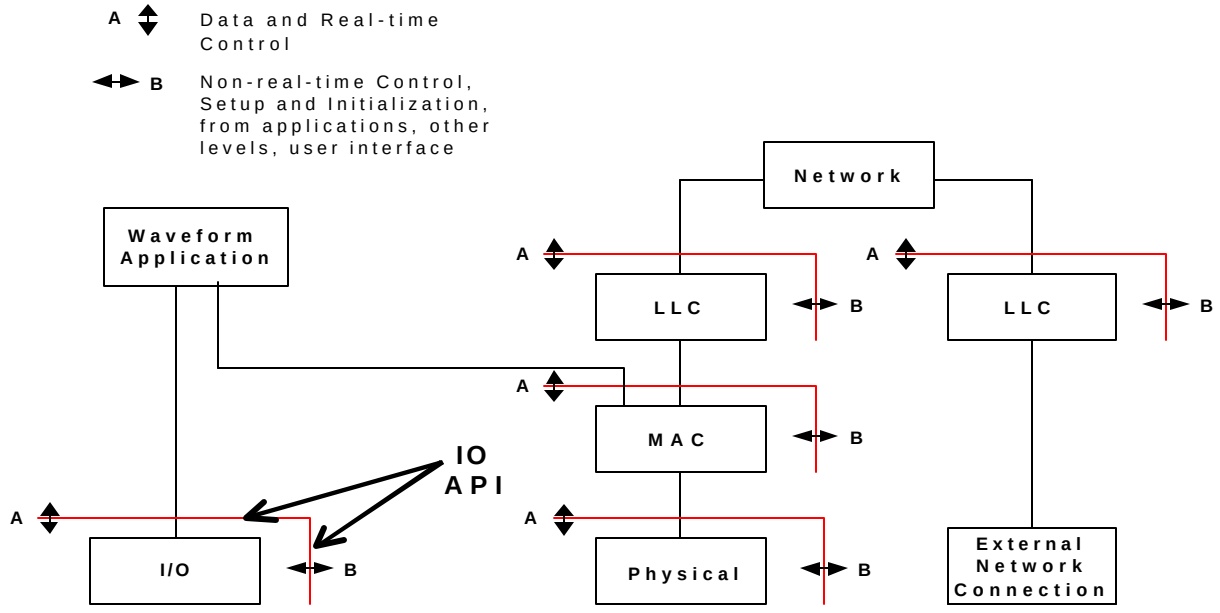

Figure 1-1. Service Definition Overview

Note: Identification and details concerning exceptions on all interfaces will be supplied in a later version of this document in accordance with SCA requirements and good engineering practice.

1.2 SERVICE LAYER DESCRIPTION.

I/O Building Blocks (BBs) specified by the SCAS I/O BB provide the fundamental structure for defining interfaces for SW resources to communicate with specific types of I/O devices. The I/O BBs are instantiated and extended by the I/O API. The SCA I/O Building Blocks are defined as:

- I/O Configuration
- I/O Control
- Audible Alerts and Alarms
- I/O Signals.

The I/O API extends BBs to provide service for audio and data devices.

1.3 MODES OF SERVICE.

There are three modes of service defined at this level for the I/O API: Transmit mode, Receive mode and Idle (or Passive Receive). Further definition of service modes is provided in later sections of this document.

1.4 SERVICE STATES.

Service states are defined in Sections 3 and 4 of this document, as required.

1.5 REFERENCED DOCUMENTS.

Document No.	Document Title
MSRC-5000SCA	Software Communications Architecture Specification
MSRC-5000API	Application Program Interface Supplement to the Software Communications Architecture Specification, Appendix H, I/O Building Block Service Definition
MSRC-5000API	Application Program Interface Supplement to the Software Communications Architecture Specification, Appendix C Generic Packet Building Block Service Definition

2 UUID.

The UUID for this API is `8aef0960-d1d3-11d4-8cc8-00104b23b8a2.`

3 SERVICES.

Two kinds of services are provided, real-time and non-real-time.

3.1 NON-REAL-TIME SERVICES.

The features of Non-Real-Time or "B" interfaces are defined in terms of services provided by the Service Provider, and the individual primitives that may flow between the Service User and Service Provider.

The non-real-time services are tabulated in Table 1 and described more fully in section 4.1.

Table 1. Cross-Reference of Non-Real-Time Services and Primitives

Service Group	Service	Primitives
Audio I/O Configuration	Configuration	DataSamplesize
		Enable - CVSD
		Version
		AudioModuleName
		AudioChanNumber
		LPC10Enable
Data I/O Configuration	Flow Control	HWFlowControl
		XonXoffControl
		None
	Asynchronous Port Configuration	dataRateInHz
		NumberOfStopBits
		ParityBit
		NumberOfDataBits
		NumberOfStartBits
	Synchronous Port Configuration	dataRateInHz

Table 1. Cross-Reference of Non-Real-Time Services and Primitives - Continued

Service Group	Service	Primitives
Audio I/O Control	Gain Control	SideToneGainIndB
		OutputGainIndB
		MicrophoneGainIndB
	AGC Control	AGCAttackTimeInmS
		AGCReleaseTimeInmS
		AGCDynamicRangeIndB
	Output Control	AudioOutputEnabled
		SidetoneEnabled
		rxTraffic
	Input Control	EnableRTS/CTS
		SetCTS
		txTraffic
	RT Status	txActive
		rxActive
Data I/O Control	Input Control	EnableRTS/CTS
		SetCTS
	RT Status	txActive
		rxActive
	Data Mode Control	DDMCActive
		ADMCActive
I/O Signals	PTT State	signalRTS
Audible Alerts And Alarms	Tones	createTone
		startTone
		stopTone
		stopAllTones
	Beeps	createBeep
		cendBeep
	Status	txActive
		rxActive

3.1.1 I/O Configuration.

I/O Configuration interfaces are obtained by instantiating the SCA I/O API Building Block ConfigurationType parameter with concrete types. AudioConfigurationType is used to realize a AudioIOConfiguration interface stereotype. And DataConfigurationType is used to realize a DataIOConfiguration interface stereotype.

3.1.1.1 AudioIOConfiguration.

This interface provides services, shown in Figure 3-1, that are used to configure a radio's audio interface to an operator. Audio devices are uniquely accessed via their object reference.

3.1.1.2 DataIOConfiguration Interface.

The DataIOConfiguration Interface provides data port configuration services. Interface details are defined by instantiating the DataIOConfiguration Interface using FlowConfigurationType and DataConfigurationType definitions, as shown in Figure 3-1. Data Ports are uniquely accessed via their object reference.

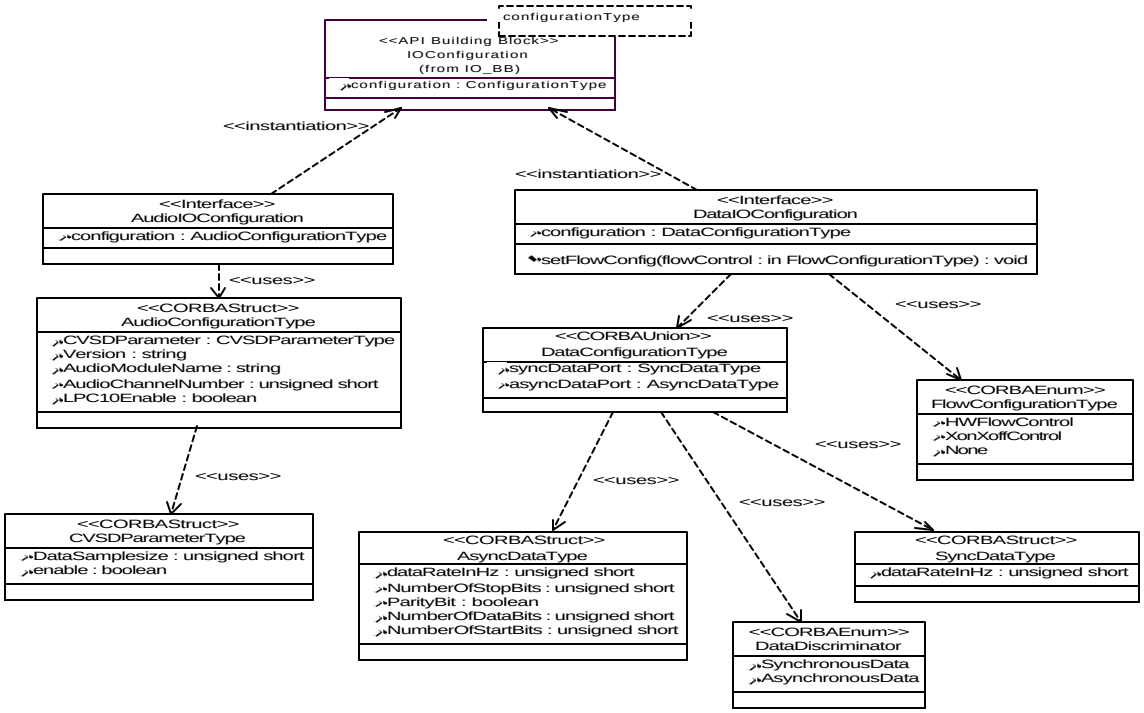

Figure 3-1. I/O Configuration

3.1.2 I/O Control Services.

The I/O Control Service Group (Figure 3-2) defines the interface for controlling the operational parameters associated with Audio and Data Devices. The SCA Audio Control Building Block is extended by instantiation of Audio and Data specific controls. The parameters listed below are implemented as Class Attributes. There are read and write operations associated with each attribute. The Service Group's operations facilitate use of RTS/CTS to provide radio push to talk

(PTT) to the Waveform API. The auto-generated "get" and "set" operations are used to set and read the state of the txActive and rxActive attributes. Except where they are defined in the building blocks, the use of attributes and their default "get"/"set" operations has been avoided since they do not support user-defined exceptions.

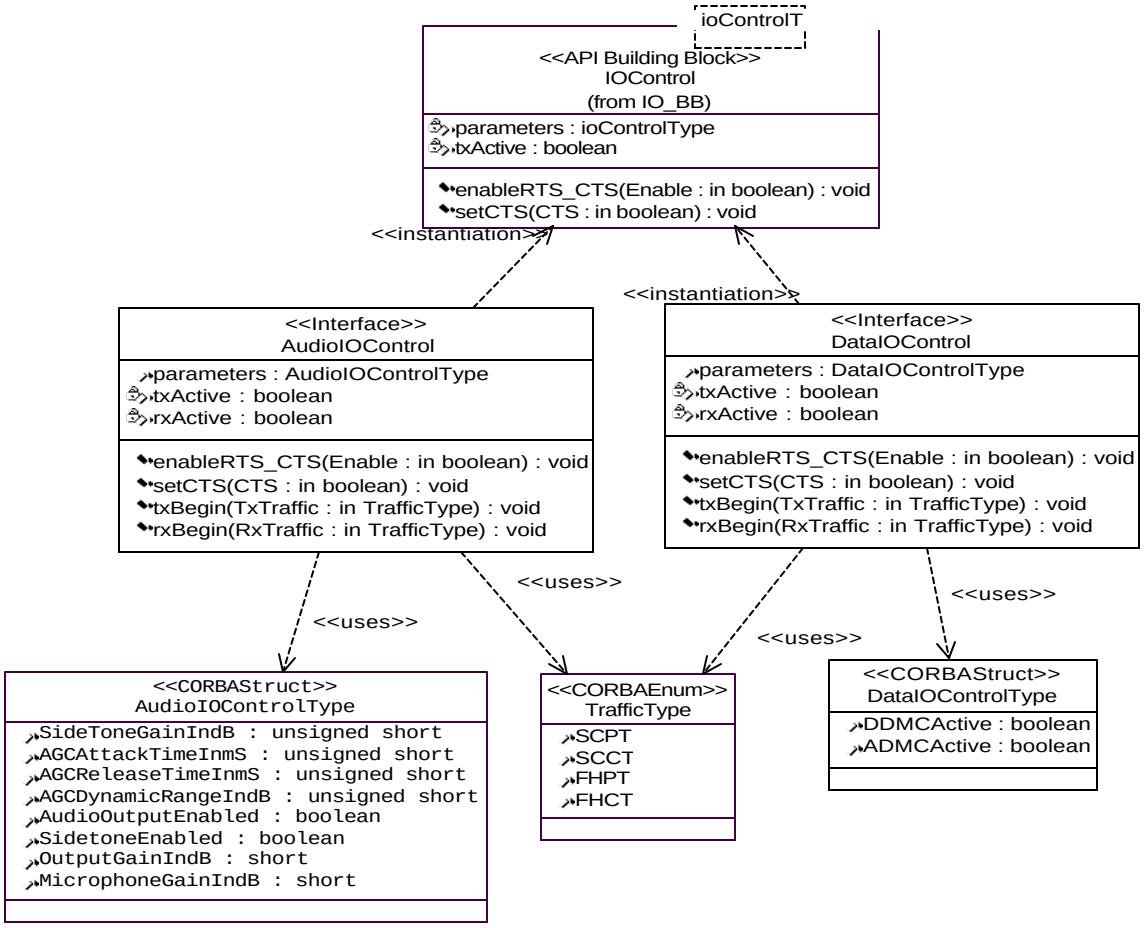

Figure 3-2. I/O Control

3.1.3 Audible Alerts and Alarms Service.

The Audible Alerts and Alarms Service Group provides sound (Tone and Beep) control interfaces. The interface details of each alert and alarm are defined by instantiating the Audio Alerts and Alarms Building Block using ToneProfileType definitions and BeepProfileType, as shown in Figure 3-3. The auto-generated "get" and "set" operations are used to set and read the state of the txActive and rxActive attribute. Except where they are defined in the building blocks, the use of attributes and their default "get"/"set" operations has been avoided since they do not support user-defined exceptions.

The Audible Alerts And Alarms Service Group provides the Service User with the capability to configure and control tones in the operator's handset that are generated by the Service Provider and txActive and rxActive services to the Audio I/O Service User. A single beep occurs each

time the sendBeep operation is invoked. Beeps are defined as a single frequency tone with an associated amplitude and duration.

A repeating tone begins when the startTone operation is invoked and ends when the stopTone operation is invoked. Repeating tones are defined as a single frequency tone with associated amplitude, and On-time and Off-time duration.

A multi-tone begins when the startTone operation is invoked and ends when the stopTone operation is invoked. Multi-tones are defined by multiple frequency and duration.

A continuous tone begins when the startTone operation is invoked and ends when the stopTone operation is invoked. They are defined as a single frequency and associated amplitude.

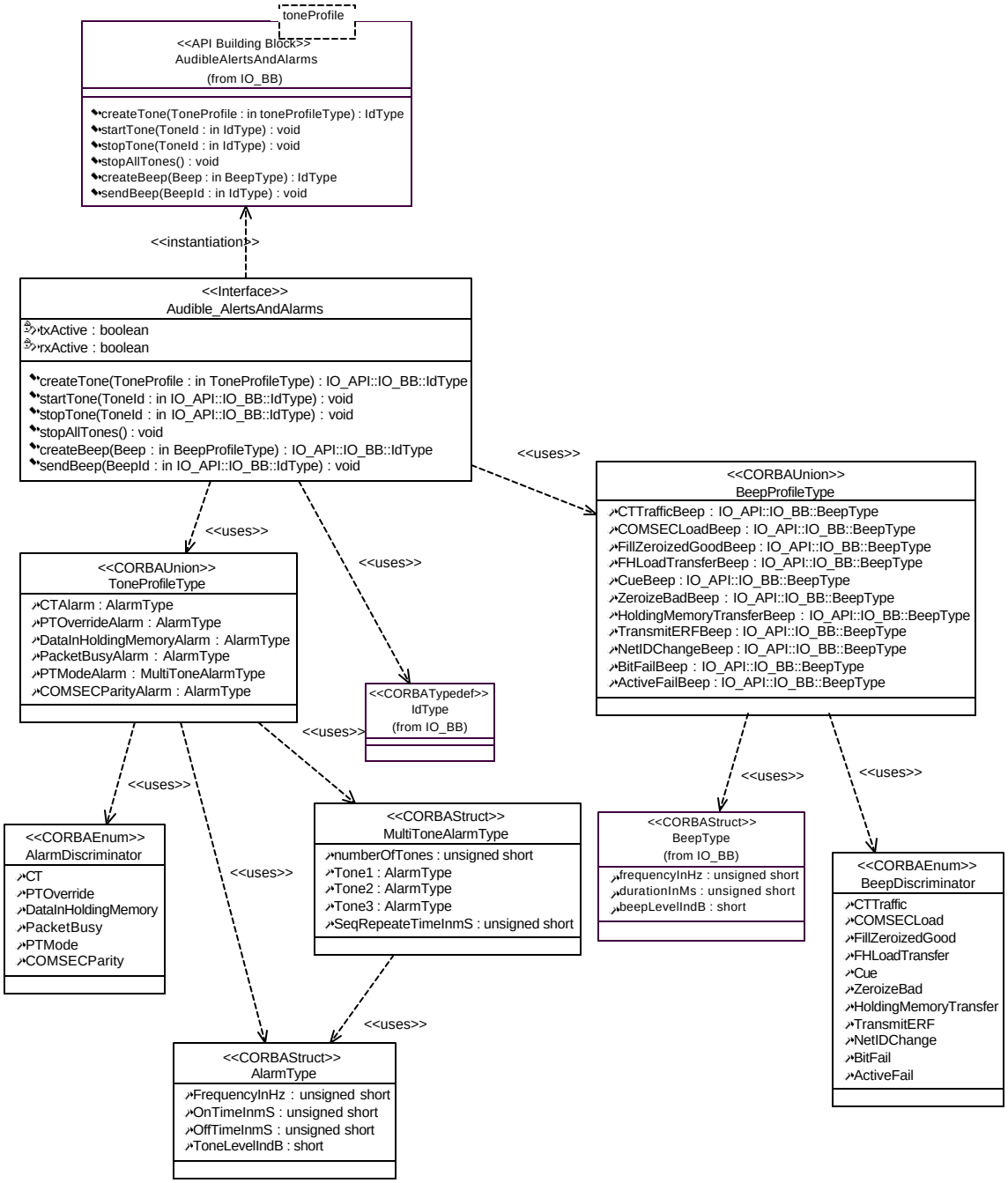

Figure 3-3. Audio Alerts and Alarms

3.1.4 I/O Signals Service.

The signalRTS interface allows the I/O Service Provider to signal the state of push-to-talk (PTT) to other JTRS layers. When PTT is active, RTS is TRUE indicating the intent to send traffic. RTS state transitions follow PTT state transitions in near real-time.

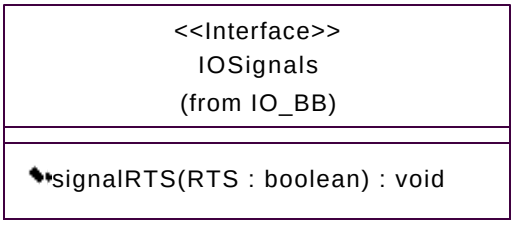

Figure 3-4. signalRTS

signalRTS is used in conjunction with Audio Control methods enableRTS/CTS and setCTS as shown in Figure 3-4 Sequence Diagram, Voice. An I/O User controls upstream flow of handset audio samples using RTS/CTS, when the enable RTS/CTS method has been invoked with boolean set to "True".

A Voice User signals a request to transmit, via handset push-to-talk (PTT), to the I/O, which forwards this request upstream by invoking the signalRTS method. When the MAC recognizes a transition of RTS to "True", it determines what action will be taken. If conditions are right, the MAC responds to the voice transmission request by invoking the setCTS method on the INFOSEC. This initializes the COMSEC function. At the completion of COMSEC initialization, a beep is issued to the handset and CTS is set to "True" indicating to the I/O that the COMSEC is ready to accept samples of handset microphone audio. Samples are then pushed upstream by the I/O as long as RTS and CTS are both "True". Upon release of handset PTT, RTS is set to "False" and upstream transfer of audio samples is terminated. When the COMSEC has processed the last audio sample, CTS is set to "False". CTS may also be set to "False" due to a COMSEC alarm condition.

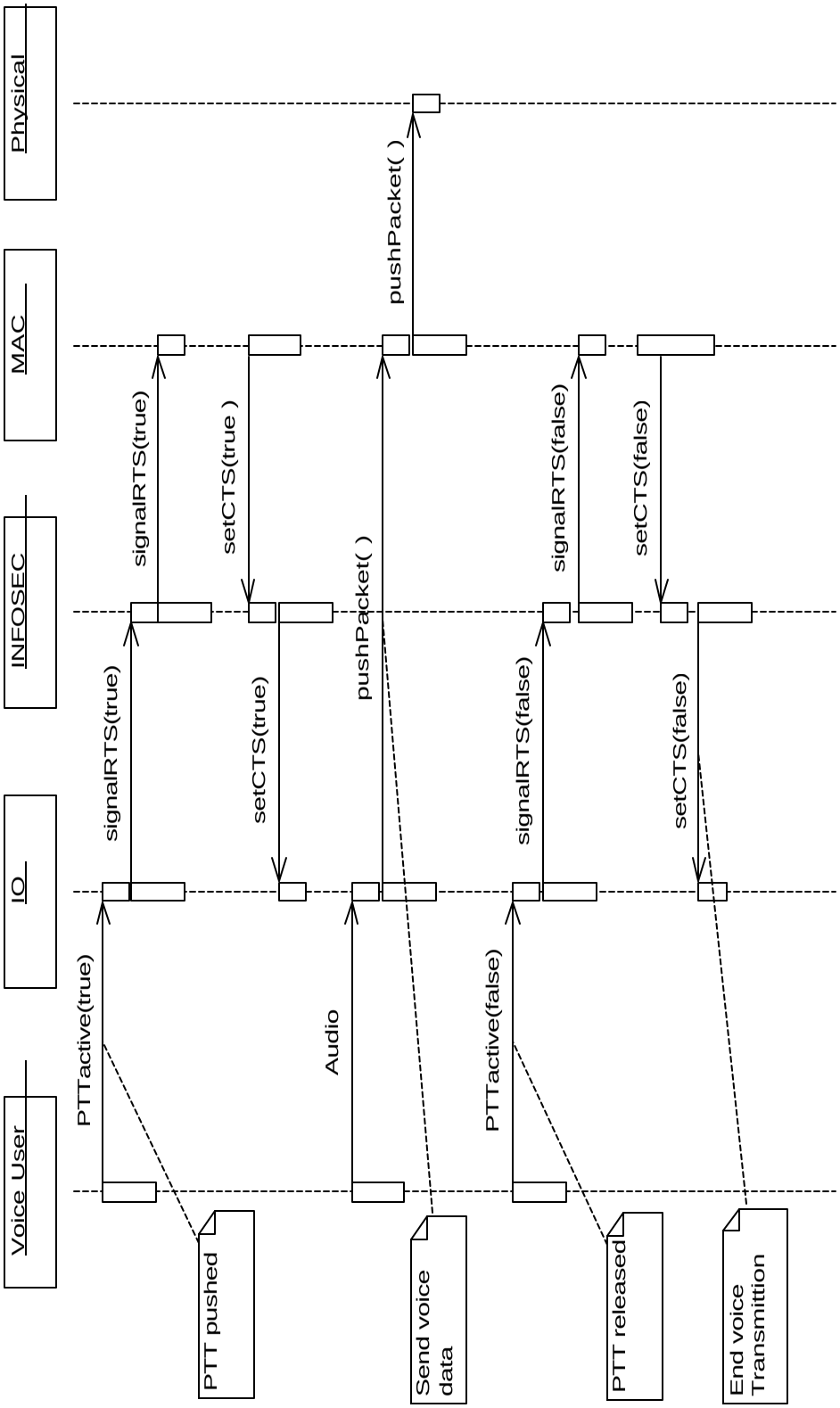

Figure 3-5. Sequence Diagram, Voice

3.2 REAL TIME SERVICES.

The features of Real-Time or "A" interfaces are defined in terms of services provided by the Service Provider, and the individual primitives that may flow between the Service User and Service Provider.

To provide upstream and downstream transfer of Audio packets between Service Users and Providers, the SCA Packet Building Block is instantiated with Payload and Control types suitable for implementing an I/O device. Figure 10-4, I/O Packet, shows the instantiation of the Packet Building Block and utilization of the Packet Signals Building Block.

Note: For waveforms that do not utilize queues, the control types should be ignored and only the pushPacket method will be used.

Real-time services are tabulated in Table 2 and described more fully in section 4.2.

Table 2. Cross-Reference of Real-Time Services and Primitives

Service Group	Service	Primitives
I/O Down Stream Provider Queue	Flow Control	MaxPayloadSize
		MinPayloadSize
		SpaceAvailable
		NumOfPriorityQueues
		SetNumOfPriorityQueues
		enableFlowControlSignals
		enableEmptySignal
	Packet	pushPacket
I/O Up Stream Provider Queue	Flow Control	MaxPayloadSize
		MinPayloadSize
		SpaceAvailable
		NumOfPriorityQueues
		SetNumOfPriorityQueues
		enableFlowControlSignals
		enableEmptySignal
	Packet	pushPacket
I/O Error Signal	Packet Data Error	SignalError

3.2.1 Packet.

I/O Packet interfaces for data transfers are based on SCA Generic Packet Building Blocks, as shown in Figure 10-4.

3.2.1.1 Flow Control.

Flow control is provided for both Service Users and Service Providers as shown in Figure 3-6 and Figure 3-7.

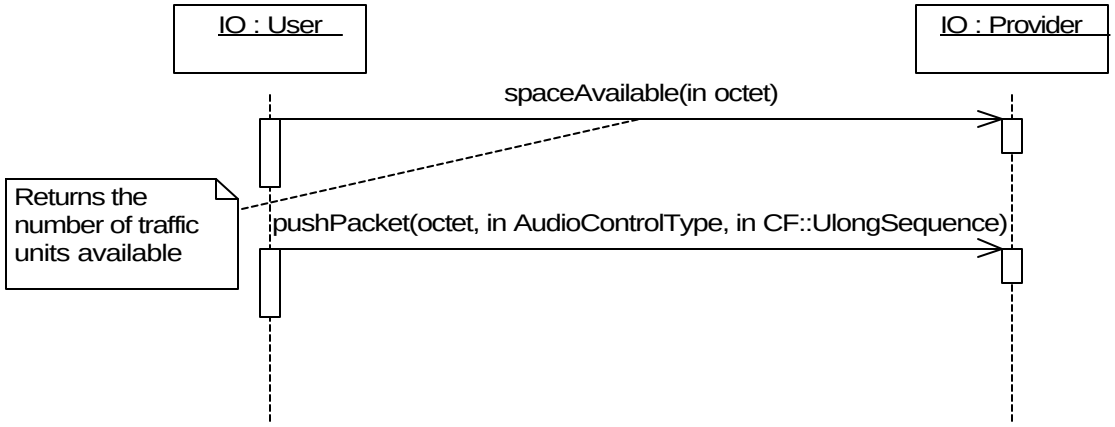

Figure 3-6. Service User Flow Control

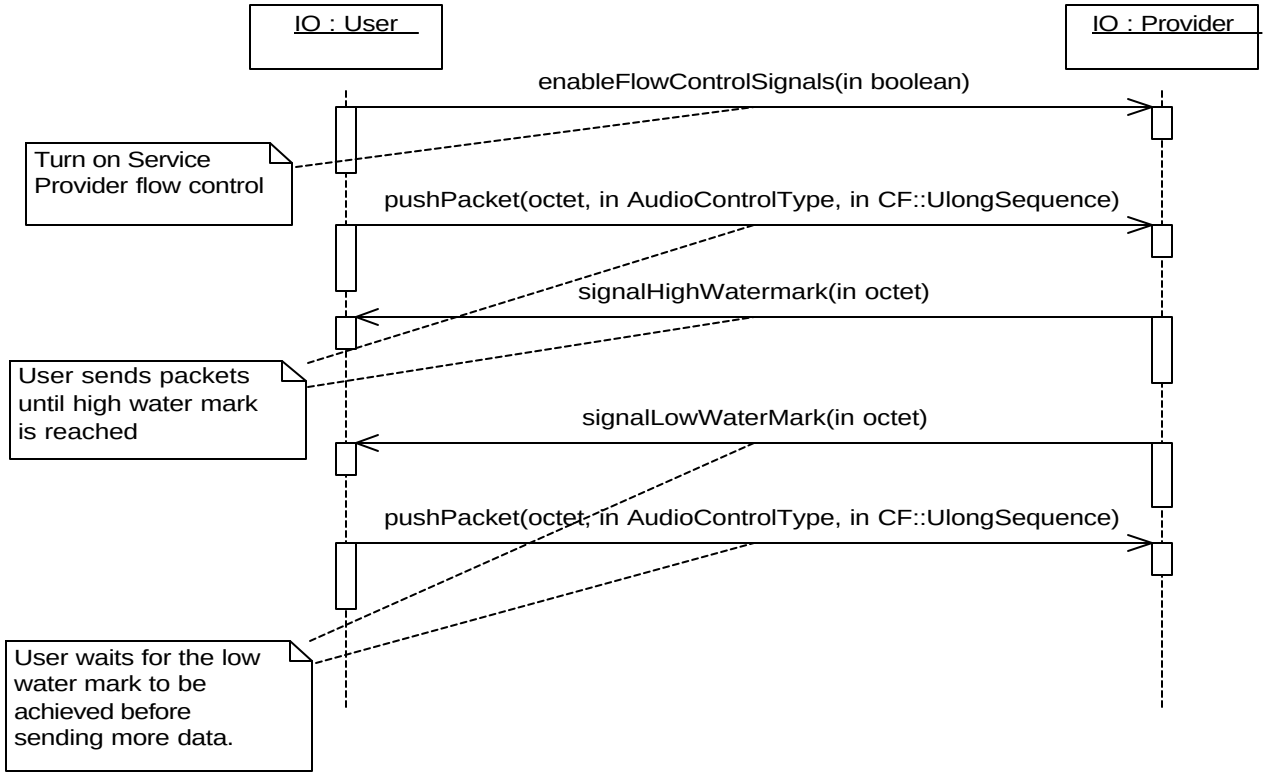

Figure 3-7. Service Provider Flow Control

3.2.1.2 Data Transfer.

pushPacket data transfer is shown in Figure 3-8..

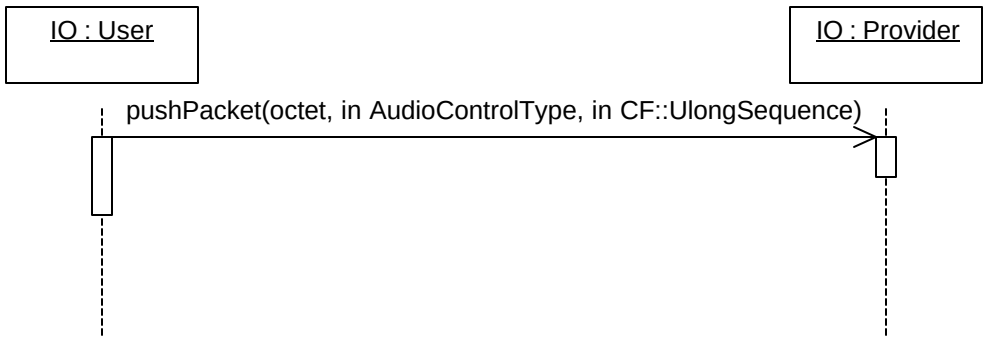

Figure 3-8. Data Transfer

3.2.1.3 Signals.

Service Providers signal all of their queues are empty as shown in Figure 3-9.

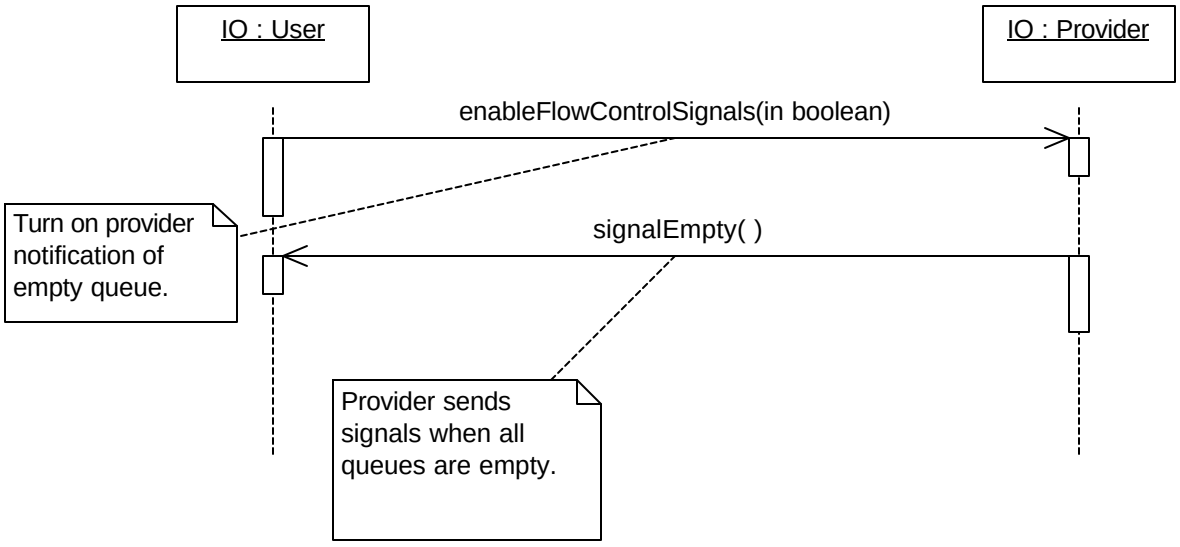

Figure 3-9. Service Provider Empty Signal

3.2.2 IOErrorSignal Service.

The IOErrorSignal Service is shown in Figure 3-10.

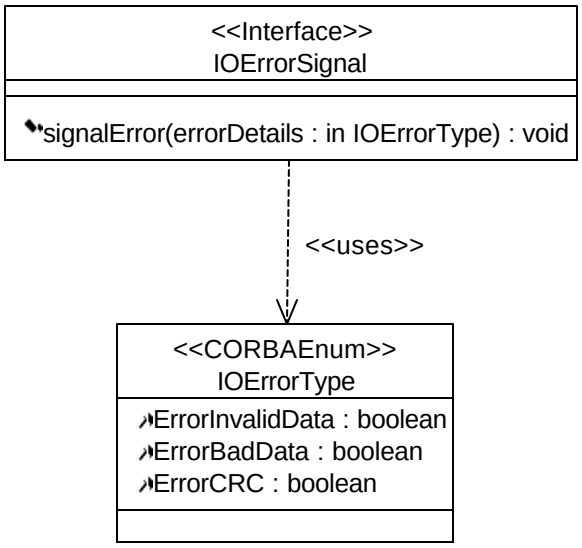

Figure 3-10. IOErrorSignal

4 SERVICE PRIMITIVES.

The I/O uses non-real-time and real-time service primitives.

Except where they are defined by a SCA Building Block, the use of attributes and their default "get"/"set" operations is avoided as they do not support user-defined exceptions.

4.1 NON-REAL-TIME PRIMITIVES.

These primitives support the "B" interface shown in Figure 1-1.

4.1.1 I/O Configuration.

The SCA I/O Building Block is instantiated twice. Once with a configuration Type of AudioIOConfiguration and again with a configuration Type DataIOConfiguration.

4.1.1.1 AudioIOConfiguration Attribute.

The AudioIOConfiguration attribute provides Service Users with the capability to configure the Service Providers audio functions.

4.1.1.1.1 Synopsis.

attribute AudioIOConfigurationType configuration

Configuration is an attribute, whose "Get" and "Set" functions are auto-generated.

4.1.1.1.2 Parameters.

configuration

This parameter has the following structure:

```
Struct AudioIOConfigurationType  {
        CVSDParameterType CVSDParameter;
        string Version;
        string AudioModuleName
        unsigned short AudioChannelNumber
        boolean LPC10Enable
};
```

CVSDParameter

This parameter allows the Service User to set/get the sample size (e.g., bits/sample) of audio data samples produced by the Service Provider.

The CVSDParameter parameter has the following structure:

```
Struct CVSDParameterType {
        Unsigned short DataSamplesize
        boolean enable
};
```

DataSamplesize

Specifies the number of data-bits per audio sample.

enable

Activates CVSD.

Version

conveys the Version Number of the Service Provider module.

AudioModuleName

conveys the Name of Service Provider module.

LPC10Enable

Tells the Audio waveform to send and receive LPC10 data.

4.1.1.1.3 State.

Any valid state as defined by the "parameters" parameter.

4.1.1.1.4 New State.

Any valid state as defined by the "parameters" parameter.

4.1.1.1.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via the "get" operation.

4.1.1.1.6 Originator.

Waveform Application.

4.1.1.1.7 Errors/Exceptions.

None.

4.1.1.2 DataIOConfiguration.

4.1.1.2.1 DataConfigurationType.

The DataConfigurationType attribute provides Service Users with the capability to configure a Service Provider’s data port to be either synchronous or asynchronous, as well as to "set"/"get" other port parameters.

4.1.1.2.1.1 Synopsis.

attribute DataConfigurationType configuration;

Configuration is an attribute, whose "Get" and "Set" functions are auto-generated.

4.1.1.2.1.2 Parameters.

configuration

This parameter specifies one of two types of configuration data, synchronous or asynchronous:

union DataConfigurationType switch(DataDiscriminator) {

case SynchronousData: SyncDataType syncDataPort;

case AsynchronousData: AsyncDataType asyncDataPort;

};

DataDiscriminator

Indicates the type of configuration to the I/O data port Service Provider:

SynchronousData	conveys configuration data is for synchronous operation.
AsynchronousData	conveys configuration data is for asynchronous operation.

syncDataPort

This parameter has the following structure:

struct SyncData_T {

unsigned short DataRateInHz;

};

dataRateInHz	indicates to the Service Provider the rate, in Hertz, of the data-clock output required to exchange MIL-STD-188-114 data with connected data terminal equipment. Data rate is provided via Resource Config/Query.

asyncDataPort

This parameter is used to configure an asynchronous data port and has the following structure:

struct AsyncDataType {

unsigned short dataRateInHz;

unsigned short NumberOfStopBits;

unsigned short ParityBit;

unsigned short NumberOfDataBits;

unsigned short NumberOfStartBits;

};

dataRateInHz	indicates the RS-232 Baud rate which the Service Provider will use to exchange data with a data terminal. Typical values are 1200, 2400, 4800, 9600. Data rate is provided via Resource Config/Query.
NumberOfStopBits	indicates the number of Stop Bits. E.g., None, 1 or 2.
ParityBit	indicates the parity configuration. E.g., 0 = None, 1 = Odd, 2 = Even, 3 = Mark, 4 = Space.
NumberOfDataBits	indicates the number of Data bits. E.g., 4, 5, 6, 7 or 8.
NumberOfStartBits	indicates the number of Start bits. E.g., 1 or 2.

4.1.1.2.1.3 State.

Any operational state.

4.1.1.2.1.4 New State.

The flow Control State of the I/O device is changed to provide the type of flow control specified.

4.1.1.2.1.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via the "get" operation.

4.1.1.2.1.6 Originator.

Waveform Application.

4.1.1.2.1.7 Errors/Exceptions.

None.

4.1.1.2.2 setFlowConfig.

The setFlowConfig service provides a Service User with the capability to configure flow control, between a connected data terminal and Service Provider.

4.1.1.2.2.1 Synopsis.

void setFlowConfig (in FlowConfigureType flowControl);

4.1.1.2.2.2 Parameters.

flowControl

Conveys the type of digital data flow control to be used between a connected data terminal and Service Provider:

HWFlowControl	indicates hardware flow control
Xon/XoffControl	indicates Xon/Xoff protocol is used for flow control
None	indicates flow control is not provided.

4.1.1.2.2.3 State.

Any operational state.

4.1.1.2.2.4 New State.

The flow Control State of the I/O device is changed to provide the type of flow control specified.

4.1.1.2.2.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via the "get" operation.

4.1.1.2.2.6 Originator.

Waveform Application.

4.1.1.2.2.7 Errors/Exceptions.

None.

4.1.2 I/O Control Services.

The following services support the IO interface to the radio operator.

4.1.2.1 Audio I/O Control.

The Audio I/O Control interface is an installation of the SCA Audio Building Block. The BB's ioControlType parameter is replaced with the AudioIOControlType parameter to provide a concrete class for audio I/O control.

4.1.2.1.1 AudioIOControlType Attribute.

4.1.2.1.1.1 Synopsis.

attribute AudioIOControlType parameters;

Parameters is an attribute whose "Get" and "Set" functions are auto-generated.

4.1.2.1.1.2 Parameters.

parameters

This parameter has the following structure:

```
struct AudioIOControlType {
        unsigned short SideToneGainIndB;
        unsigned short AGCAttackTimeInmS;
        unsigned short AGCReleaseTimeInmS;
        unsigned short AGCDynamicRangeIndB;
        boolean AudioOutputEnabled;
        boolean SidetoneEnabled;
        short OutputGainIndB;
        short MicrophoneGainIndB;
};
```

SideToneGainIndB	sets the Sidetone audio level delivered to the handset with respect to the received audio level, in dB. (e.g., -20dB specifies the Sidetone audio level will be 20 dB below the received audio level). This parameter may be dynamically set during operation.
AGCAttackTimeInmS	sets the Attack Time, in milliseconds, of the AGC applied to incoming audio from a microphone. This parameter may NOT be dynamically set during operation.
AGCReleaseTimeInmS	sets the Release Time, in milliseconds, of the AGC applied to incoming audio from a microphone. This parameter may NOT be dynamically set during operation.
AGCDynamicRangeIndB	sets the dynamic range of the AGC applied to incoming audio from a microphone to keep it within the range of the

	A/D converter that is sampling the input audio. This parameter may NOT be dynamically set during operation.
AudioOutputEnabled	Boolean TRUE enables audio output; FALSE disables audio output. This parameter may be dynamically set during operation.
SidetoneEnabled	Boolean TRUE enables audio output; FALSE disables audio output. This is used to gate sidetone audio ON during voice traffic and OFF during data traffic. This parameter may be dynamically set during operation.
OutputGainIndB	sets the level, in dB, of the audio output to a handset or speaker during receive and is used to provide volume control. This parameter may be dynamically set during operation.
MicrophoneGainIndB	sets the level in dB of the gain applied to audio from the microphone to adjust between Normal and Whisper modes. This parameter may be dynamically set during operation.

4.1.2.1.1.3 State.

Any operational state.

4.1.2.1.1.4 New State.

For all current and future receive operations, the Audio device implements the commanded states.

4.1.2.1.1.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via the "get" operation.

4.1.2.1.1.6 Originator.

Waveform Application.

4.1.2.1.1.7 Errors/Exceptions.

None.

4.1.2.1.2 txActive Attribute.

This attribute allows the Service User to get the Transmit State of the Audio I/O Device.

4.1.2.1.2.1 Synopsis.

readonly attribute boolean txActive

txActive is an attribute whose "Get" function is auto-generated.

4.1.2.1.2.2 Parameters.

None.

4.1.2.1.2.3 State.

Any operational state.

4.1.2.1.2.4 New State.

None.

4.1.2.1.2.5 Response.

The auto generated Get function returns boolean values:

txActive = True

Indicates the I/O device is transmitting data (i.e., pushing packets downstream).

txActive = False

Indicates the I/O device is not transmitting data.

4.1.2.1.2.6 Originator.

Waveform Application.

4.1.2.1.2.7 Errors/Exceptions.

None.

4.1.2.1.3 rxActive.

This attribute allows the Service User to get the Receive State of the Audio I/O Device.

4.1.2.1.3.1 Synopsis.

readonly attribute boolean rxActive

rxActive is an attribute whose "Get" function is auto-generated.

4.1.2.1.3.2 Parameters.

None.

4.1.2.1.3.3 State.

Any operational state.

4.1.2.1.3.4 New State.

None.

4.1.2.1.3.5 Response.

The auto generated Get function returns boolean values:

rxActive = True

Indicates the I/O device is receiving data (i.e., pushing packets upstream).

rxActive = False

Indicates the I/O device is not receiving data.

4.1.2.1.3.6 Originator.

Waveform Application.

4.1.2.1.3.7 Errors/Exceptions.

None.

4.1.2.1.4 enableRTS/CTS Service.

This attribute provides Service Users with the ability to convey to the Service Provider the desired flow control method to be used when pushing I/O data upstream via pushPacket.

When RTS/CTS is enabled, the Service Provider pushes data upstream when Request To Send (RTS) is active (indicating handset PTT is active) and Clear To Send (CTS) is active (indicating the modem is ready to accept I/O data). When disabled, the Service Provider will push packets upstream when handset PTT is active, regardless of the state of CTS. The normal mode of SINCGARS operation is to enable RTS/CTS. During Single Channel Plain Text operation, RTS/CTS may be disabled.

4.1.2.1.4.1 Synopsis.

oneway void enableRTS/CTS (in boolean Enable);

4.1.2.1.4.2 Parameters.

Enable = True

Conveys to the downstream data provider to push I/O data samples upstream under the control of the RTS and CTS signals.

Enable = False

Conveys to the downstream data provider to unilaterally push I/O data samples upstream (e.g., ignore RTS and CTS).

4.1.2.1.4.3 State.

Any operational state.

4.1.2.1.4.4 New State.

For all current and future transmit operations, the data device implements the commanded flow control configuration.

If *Enable* is "TRUE", I/O samples are pushed upstream under the control of RTS or CTS.

If *Enable* is "FALSE", I/O samples are pushed upstream and RTS and CTS are ignored.

4.1.2.1.4.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via the "get" operation.

4.1.2.1.4.6 Originator.

Waveform Application.

4.1.2.1.4.7 Errors/Exceptions.

None.

4.1.2.1.5 setCTS Service.

The Set Clear To Send (CTS) operation provides a Service User the ability to control the upstream flow of microphone audio samples or data samples from the Service Provider.

4.1.2.1.5.1 Synopsis.

oneway void setCTS (in boolean CTS);

4.1.2.1.5.2 Parameters.

CTS = True

Conveys to the downstream data provider the upstream data user is ready to accept data via the pushPacket transfer mechanism.

CTS = False

Conveys to the downstream data provider the upstream data user is not ready to accept data via the pushPacket transfer mechanism.

4.1.2.1.5.3 State.

Any operational state.

4.1.2.1.5.4 New State.

For all current and future transmit operations, the data device implements the commanded CTS state.

If RTS is "TRUE" and CTS is "TRUE", I/O samples are pushed upstream until either RTS or CTS becomes "FALSE".

4.1.2.1.5.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via a "get" operation.

4.1.2.1.5.6 Originator.

Waveform Application.

4.1.2.1.5.7 Errors/Exceptions.

None.

4.1.2.1.6 txBegin.

The txBegin attribute is used to configure the audio path for voice transmission. Based on the transmitter traffic mode, analog audio is routed through the CVSD or the CVSD is bypassed (bypassed in SCPT).

4.1.2.1.6.1 Synopsis.

void txBegin (in TrafficType TxTraffic);

4.1.2.1.6.2 Parameters.

TxTraffic

Indicates the traffic mode of the radio:

SCPT	indicates Single Channel Plain Text
SCCT	indicates Single Channel Cipher Text
FHPT	indicates Frequency Hop Plain Text
FHCT	indicates Frequency Hop Cipher Text

4.1.2.1.6.3 State.

Any operational state.

4.1.2.1.6.4 New State.

For all current and future transmit operations, the data device implements the commanded audio path configuration.

4.1.2.1.6.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via the "get" operation.

4.1.2.1.6.6 Originator.

Waveform Application.

4.1.2.1.6.7 Errors/Exceptions.

None defined.

4.1.2.1.7 rxBegin.

The rxBegin attribute is used to configure the audio path for voice reception. Based on the receiver traffic mode, analog audio is routed through the CVSD or the CVSD is bypassed (bypassed in SCPT).

4.1.2.1.7.1 Synopsis.

void rxBegin (in TrafficType RxTraffic);

4.1.2.1.7.2 Parameters.

RxTraffic

Indicates the traffic mode of the radio:

SCPT	indicates Single Channel Plain Text
SCCT	indicates Single Channel Cipher Text
FHPT	indicates Frequency Hop Plain Text
FHCT	indicates Frequency Hop Cipher Text

4.1.2.1.7.3 State.

Any operational state.

4.1.2.1.7.4 New State.

For all current and future receive operations, the data device implements the commanded audio path configuration.

4.1.2.1.7.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via the "get" operation.

4.1.2.1.7.6 Originator.

Waveform Application.

4.1.2.1.7.7 Errors/Exceptions.

None defined.

4.1.2.2 Data I/O Control.

The Data I/O Control interface is an instantiation of the SCA Audio Building Block. The BB's ioControlType parameter is replaced with the DataIOControl parameter to provide a concrete class for audio I/O control.

4.1.2.2.1 Data I/O Control Type Attribute.

4.1.2.2.1.1 Synopsis.

attribute DataIOControlType parameters

parameters is an attribute whose "Get" function is auto-generated.

4.1.2.2.1.2 Parameters.

Not applicable.

4.1.2.2.1.3 State.

Any operational state.

4.1.2.2.1.4 New State.

None.

4.1.2.2.1.5 Response.

The auto generated "Get" function returns the current boolean values for the parameter.

DataIOControlType

The DataIOCntl_T parameter has the following structure:

```
struct DataIOControlType {
        boolean DDMCActive;
        boolean ADMCActive;
};
```

DDMCActive	*TRUE* indicates the interface to the connected data terminal equipment is MIL-STD-188-114 digital.
ADMCActive	*TRUE* indicates the interface to the connected data terminal equipment is analog.

4.1.2.2.1.6 Originator.

Waveform Application.

4.1.2.2.1.7 Errors/Exceptions.

None.

4.1.2.2.2 txActive Attribute.

This attribute provides the ability to fetch the state of the I/O device’s transmitting function.

4.1.2.2.2.1 Synopsis.

readonly attribute boolean txActive;

4.1.2.2.2.2 Parameters.

None

4.1.2.2.2.3 State.

Any operational state.

4.1.2.2.2.4 New State.

None.

4.1.2.2.2.5 Response.

The auto generated "Get" function returns boolean values:

txActive = True

Indicates the I/O device is transmitting data.

txActive = False

Indicates the I/O device is not transmitting data.

4.1.2.2.2.6 Originator.

Waveform Application.

4.1.2.2.2.7 Errors/Exceptions.

None.

4.1.2.2.3 rxActive.

4.1.2.2.3.1 Synopsis.

readonly attribute boolean rxActive;

4.1.2.2.3.2 Parameters.

None.

4.1.2.2.3.3 State.

Any operational state.

4.1.2.2.3.4 New State.

None.

4.1.2.2.3.5 Response.

The auto generated "Get" function returns boolean values:

rxActive = True

Indicates the I/O device is receiving data.

rxActive = False

Indicates the I/O device is not receiving data.

4.1.2.2.3.6 Originator.

Waveform Application.

4.1.2.2.3.7 Errors/Exceptions.

None.

4.1.2.2.4 enableRTS/CTS Service.

This attribute provides Service Users with the ability to convey to the Service Provider the desired flow control method to be used when pushing I/O data upstream via pushPacket.

4.1.2.2.4.1 Synopsis.

oneway void enableRTS/CTS (in boolean Enable);

4.1.2.2.4.2 Parameters.

Enable = True

Conveys to the downstream data provider to push I/O data samples upstream under the control of the RTS and CTS signals.

Enable = False

Conveys to the downstream data provider to unilaterally push I/O data samples upstream (e.g., ignore RTS and CTS).

4.1.2.2.4.3 State.

Any operational state.

4.1.2.2.4.4 New State.

If Enable is "TRUE", I/O samples are pushed upstream under the control of RTS or CTS.

If Enable is "FALSE", I/O samples are pushed upstream and RTS and CTS are ignored.

4.1.2.2.4.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via the "get" operation.

4.1.2.2.4.6 Originator.

Waveform Application.

4.1.2.2.4.7 Errors/Exceptions.

None.

4.1.2.2.5 SetCTS Service.

The Set Clear To Send operation provides the ability to control the upstream flow of microphone audio samples or data samples.

4.1.2.2.5.1 Synopsis.

void setCTS(CTS : in boolean).

4.1.2.2.5.2 Parameters.

CTS = True

Conveys to the downstream data provider the upstream data user is ready to accept data via the pushPacket transfer mechanism.

CTS = False

Conveys to the downstream data provider the upstream data user is not ready to accept data via the pushPacket transfer mechanism.

4.1.2.2.5.3 State.

Any operational state.

4.1.2.2.5.4 New State.

If RTS is "TRUE" and CTS is "TRUE", I/O samples are pushed upstream until either RTS or CTS becomes "FALSE".

4.1.2.2.5.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via a "get" operation.

4.1.2.2.5.6 Originator.

Waveform Application.

4.1.2.2.5.7 Errors/Exceptions.

None.

4.1.2.2.6 txBegin.

The txBegin attribute is used to configure the data path for data transmission. Analog data is routed differently when the TxTraffic mode is SCPT. For all other traffic modes, data is digitized.

4.1.2.2.6.1 Synopsis.

void txBegin (in TrafficType TxTraffic);

4.1.2.2.6.2 Parameters.

TxTraffic

Indicates the traffic mode of the radio:

SCPT	indicates Single Channel Plain Text
SCCT	indicates Single Channel Cipher Text
FHPT	indicates Frequency Hop Plain Text
FHCT	indicates Frequency Hop Cipher Text

4.1.2.2.6.3 State.

Any operational state.

4.1.2.2.6.4 New State.

For all current and future transmit operations, the data device implements the commanded audio path configuration.

4.1.2.2.6.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via the "get" operation.

4.1.2.2.6.6 Originator.

Waveform Application.

4.1.2.2.6.7 Errors/Exceptions.

None.

4.1.2.2.7 rxBegin.

The rxBegin attribute is used to configure the data path for data transmission. Analog data is routed differently when the RxTraffic mode is SCPT. For all other traffic modes, data is digitized.

4.1.2.2.7.1 Synopsis.

void rxBegin (in Traffic_T RxTraffic);

4.1.2.2.7.2 Parameters.

RxTraffic

Indicates the traffic mode of the radio:

SCPT	indicates Single Channel Plain Text
SCCT	indicates Single Channel Cipher Text
FHPT	indicates Frequency Hop Plain Text
FHCT	indicates Frequency Hop Cipher Text

4.1.2.2.7.3 State.

Any operational state.

4.1.2.2.7.4 New State.

For all current and future receive operations, the data device implements the commanded audio path configuration.

4.1.2.2.7.5 Response.

It is the Service User's responsibility to verify the parameters were "set" via the "get" operation.

4.1.2.2.7.6 Originator.

Waveform Application.

4.1.2.2.7.7 Errors/Exceptions.

None.

4.1.3 Audible Alerts and Alarms Service.

4.1.3.1 createTone Service.

The createTone Service provides the Service User with the capability to specify to the Service Provider a set of tone profiles, each having a unique tone identifier. Once a tone profile is created, tone generation can be started/stopped using the unique tone identifier. Two types of tones are supported: single-tone and multi-tone. Tone generation is started by the startTone service and continues until stopped by the stopTone or stopAllTones services.

4.1.3.1.1 Synopsis.

IO_BB::IdType createTone (in ToneProfileType ToneProfile);

ToneProfile

identifies one of several types of tone profiles. This is necessary because different applications generate different types of tones (i.e., – single tone, alternating tone, multiple tones, etc.).

```
union ToneProfileType switch(AlarmDiscriminator) {
        case CT: AlarmType CTAlarm;
        case PTOverride: AlarmType PTOverrideAlarm;
        case DataInHoldingMemory: AlarmType DataInHoldingMemoryAlarm;
        case PacketBusy: AlarmType PacketBusyAlarm;
        case PTMode: MultiToneAlarmType PTModeAlarm;
        case COMSECParity: AlarmType COMSECParityAlarm;
        };
```

AlarmDiscriminator

Indicates the specific parameter that specifies the type of profile to be created.

CT	specifies the CTAlarm parameter
PTOverride	specifies the PTOverrideAlarm parameter
DataInHoldingMemory	specifies the DataInHoldingMemoryAlarm parameter
PacketBusy	specifies the PacketBusyAlarm parameter
PTMode	specifies the PTModeAlarm parameter
COMSECParity	specifies the COMSECParityAlarm parameter

CTAlarm

This parameter specifies the Cipher Text alarm tone profile and has the following structure:

```
struct AlarmType {
        unsigned short FreqencyInHz;
        unsigned short OnTimeInmS;
```

```
        unsigned short OffTimeInmS;
        short ToneLevelIndB;
};
```

FrequencyInHz

Specifies the tone frequency in Hertz

OnTimeInmS

Specifies the time the tone in audible (On time) in milliseconds.

OffTimeInmS

Specifies the time the tone in inaudible (Off time) in milliseconds.

ToneLvlndB

Specifies the tone level with respect to the received audio level, in dB (e.g., -20 dB specifies the alarm tone will be 20 dB below receive audio).

PTOverrideAlarm

This parameter specifies the Plain Text Override alarm tone profile and has the same structure as the *CTAlarm* parameter.

DataInHoldingMemoryAlarm

This parameter specifies the Data In Holding Memory alarm tone profile and has the same structure as the *CTAlarm* parameter.

PacketBusyAlarm parameter

This parameter specifies the Packet Busy alarm tone profile and has the same structure as the *CTAlarm* parameter.

PTModeAlarm

specifies the Plain Text Mode alarm tone, which is a multi-tone alarm. Initially, Tone 1 is generated upon receipt of a startTone command. Subsequent occurrences of Tone 1 occur each time the Sequence Repeat Time lapses. Tone 2 (if specified) is generated at the completion of Tone 1. Tone 3 (if specified) is generated at the completion of Tone 2. The sequence of tones is repeated until a stopTone or stopAllTones command is received.

The PTModeAlarm parameter has the following structure:

```
struct MultiToneAlarmType {
        unsigned short numberOfTones;
        AlarmType Tone1;
        AlarmType Tone2;
        AlarmType Tone3;
        unsigned short SeqRepeateTimeInmS;
};
```

NumberOfTones

Specifies the number of tones in a tone sequence: 1, 2 or 3.

Tone1

Specifies the frequency, On-time and Off-time of the first tone in a sequence.

Tone2

Specifies the frequency, On-time and Off-time of the second tone in a sequence.

Tone3

Specifies the frequency, On-time and Off-time of the third tone in a sequence.

SeqRepeateTimeInmS

Specifies the time interval, in milliseconds, between the start of each occurrence of Tone 1.

COMSECParityAlarm

This parameter specifies the COMSEC Parity alarm tone profile and has the same structure as the *CTAlarm* parameter.

4.1.3.1.2 State.

Any operational state.

4.1.3.1.3 New State.

The audio device adds the newly created tone to the tone database.

4.1.3.1.4 Response.

IO_BB::IdType

Upon creation of a Tone profile, the device will return a non-zero ID that is unique with respect to all beep and tone profile IDs.

4.1.3.1.5 Originator.

Waveform application.

4.1.3.1.6 Errors/Exceptions.

IO_BB::IdType

A value of zero is returned if the tone profile could not be created.

4.1.3.2 startTone Service.

startTone provides a Service User the ability to start Service Provider generation of a tone whose profile was previously defined via the createTone operation. A ToneId is used to specify the

profile of the tone to be started. Multiple tones may be started and each will continue until stopped.

4.1.3.2.1 Synopsis.

oneway void startTone (in IO_BB::IdType ToneId);

4.1.3.2.2 Parameters.

ToneId

identifies the tone profile of the tone to be generated.

4.1.3.2.3 State.

Any operational state.

4.1.3.2.4 New State.

The Audio device begins generation of the selected tone, if not already started, which continues until a stopTone command is sent to the Service Provider.

4.1.3.2.5 Response.

None.

4.1.3.2.6 Originator.

Waveform application.

4.1.3.2.7 Errors/Exceptions.

None.

4.1.3.3 stopTone Service.

stopTone provides the Service User the ability to stop Service Provider generation of a previously started tone. A ToneId is used to specify the tone to be stopped.

4.1.3.3.1 Synopsis.

oneway void stopTone (in IO_BB::IdType ToneId);

4.1.3.3.2 Parameters.

ToneId

identifies the profile of the tone being generated, which invocation of StopTone will terminate.

4.1.3.3.3 State.

Any operational state.

4.1.3.3.4 New State.

The Audio device stops generation of the selected tone, if not already stopped.

4.1.3.3.5 Response.

None.

4.1.3.3.6 Originator.

Waveform application.

4.1.3.3.7 Errors/Exceptions.

None.

4.1.3.4 stopAllTones Service.

stopAllTones provides the Service User the ability to stop Service Provider generation of all previously started tones.

4.1.3.4.1 Synopsis.

oneway void stopAllTones ();

4.1.3.4.2 Parameters.

None.

4.1.3.4.3 State.

Any operational state.

4.1.3.4.4 New State.

The Audio device stops generation of all tones.

4.1.3.4.5 Response.

None.

4.1.3.4.6 Originator.

Waveform application.

4.1.3.4.7 Errors/Exceptions.

None.

4.1.3.5 createBeep Service.

The createBeep Service provides a Service User the capability to request a Service Provider to create a Beep profile. When the profile has been created, the Service Provider returns a unique ID for the profile. Profiles need not be unique. However, their IDs must be unique. If a profile is not created, an ID of zero is returned. A set of Beep profiles can be created via multiple invocations of createBeep. Each Beep profile is comprised of a tone frequency, amplitude and duration. Beep generation is initiated by the sendBeep service and only one beep occurs per invocation.

4.1.3.5.1 Synopsis.

IO_BB::IdType createBeep (in BeepProfileType Beep);

4.1.3.5.2 Parameters.

Beep

identifies the type of beep profile to be created (e.g., COMSEC, CUE Alert, etc.).

Beep has the following structure:

```
union BeepProfileType  switch(BeepDiscriminator) {
        case CTTraffic: IO_BB::BeepType CTTrafficBeep;
        case COMSECLoad: IO_BB::BeepType COMSECLoadBeep;
        case FillZeroizeGood: IO_BB::BeepType FillZeroizeGoodBeep;
        case FHLoadTransfer: IO_BB::BeepType FHLoadTransferBeep;
        case Cue: IO_BB::BeepType CueBeep;
        case ZeroizeBad: IO_BB::BeepType ZeroizeBadBeep;
        case HoldingMemoryTransfer: IO_BB::BeepType HoldingMemoryTransferBeep;
        case TransmitERF: IO_BB::BeepType TransmitERFBeep;
        case NetIDChange: IO_BB::BeepType NetIDChangeBeep;
        case BitFail: IO_BB::BeepType BitFailBeep;
        case ActiveFail: IO_BB::BeepType ActiveFailBeep;
        };
```

BeepDiscriminator

Indicates the parameter to be created by the Service Provider, which will define a specific type Beep profile:

CTTraffic	selects the CTTrafficBeep parameter
COMSECLoad	selects the COMSECLoadBeep parameter
FillZeroizeGood	selects the FillZeroizeGoodBeep parameter
FHLoadTransfer	selects the FHLoadTransferBeep parameter
Cue	selects the CueBeep parameter
ZeroizeBad	selects the ZeroizeBadBeep parameter
HoldingMemoryTransfer	selects the HoldingMemoryTransferBeep parameter
TransmitERF	selects the TransmitERFBeep parameter
NetIDChange	selects the NetIDChangeBeep parameter
BitFail	selects the BitFailBeep parameter
ActiveFail	selects the ActiveFailBeep parameter

CTTrafficBeep

This parameter specifies the CT Traffic Beep tone profile and has the following structure, which is inherited from the API IO Building Block:

```
struct BeepType {
        unsigned short frequencyInHz;
        unsigned short durationInmS;
        short beepLevelndB;
};
```

COMSECLoadBeep

This parameter specifies the COMSEC Load Beep tone profile and has the same structure as the *CT TrafficBeep* parameter.

FillZeroizeGoodBeep

This parameter specifies the Fill Zeroize Good Beep tone profile and has the same structure as the *CT TrafficBeep* parameter.

FHLoadTransferBeep

This parameter specifies the FH Load Transfer Beep tone profile and has the same structure as the *CT TrafficBeep* parameter.

CueBeep

This parameter specifies the Cue Beep tone profile and has the same structure as the *CT TrafficBeep* parameter.

ZeroizeBadBeep

This parameter specifies the Zeroize Bad Beep tone profile and has the same structure as the *CT TrafficBeep* parameter.

HoldingMemoryTransferBeep

This parameter specifies the Holding Memory Transfer Beep tone profile and has the same structure as the *CT TrafficBeep* parameter.

TransmitERFBeep

This parameter specifies the Transmit ERF Beep tone profile and has the same structure as the *CT TrafficBeep* parameter.

NetIDChangeBeep

This parameter specifies the Net ID Change Beep tone profile and has the same structure as the *CT TrafficBeep* parameter.

BitFailBeep

This parameter specifies the Bit Fail Beep tone profile and has the same structure as the *CT TrafficBeep* parameter.

ActiveFailBeep

This parameter specifies the ActiveFail Beep tone profile and has the same structure as the *CT TrafficBeep* parameter.

4.1.3.5.3 State.

Any current operational state.

4.1.3.5.4 New State.

The audio device outputs the selected beep frequency and audio level to the operator’s audio device for the duration defined during the CreateBeep operation.

4.1.3.5.5 Response.

IO_BB::IdType

Upon creation of a Beep profile, the device will return a non-zero ID that is unique with respect to all beep and tone profile IDs.

4.1.3.5.6 Originator.

Waveform application.

4.1.3.5.7 Errors/Exceptions.

IO_BB::IdType

A value of zero is returned if the beep profile could not be created.

4.1.3.6 sendBeep Service.

sendBeep provides a Service User the ability to command a Service Provider to generate and output to a handset or speaker a Beep using a predefined Beep profile. BeepId is used specify the Beep Profile. One and only one Beep is generated/output per invocation of sendBeep.

4.1.3.6.1 Synopsis.

oneway void sendBeep (in IO_BB::IdType BeepId);

4.1.3.6.2 Parameters.

BeepId

identifies the profile of the Beep that invocation of SendBeep will cause the Service Provider to generate and output to the operator's audio device.

4.1.3.6.3 State.

Any operational state.

4.1.3.6.4 New State.

The Audio device begins generation of the selected beep, if not already started.

4.1.3.6.5 Response.

None.

4.1.3.6.6 Originator.

Waveform application.

4.1.3.6.7 Errors/Exceptions.

None.

4.1.4 I/O Signals.

4.1.4.1 signalRTS Service.

This signaling operation provides the ability to signal a transmit request (PTT active) to transmit -data Service Providers.

4.1.4.1.1 Synopsis.

void signalRTS(in boolean RTS)

4.1.4.1.2 Parameters.

RTS = True

signals PTT is active, indicating the operator intends to transmit traffic.

RTS = False

signals PTT is inactive.

4.1.4.1.3 State.

Any operational state.

4.1.4.1.4 New State.

None.

4.1.4.1.5 Response.

None.

4.1.4.1.6 Originator.

I/O device.

4.1.4.1.7 Errors/Exceptions.

None.

4.2 REAL-TIME PRIMITIVES.

These primitives support the "A" interface shown in Figure 1-1 and provide separate real-time interfaces for an I/O Downstream Provider and for an I/O Upstream User.

4.2.1 Queue Services.

These are common services provided by both the I/O Downstream Provider Queue and the I/O Upstream User Queue.

4.2.1.1 spaceAvailable.

This operation provides the Service User the ability to poll the Service Provider to determine the amount of queue space available, in ‘elements’, for a given queue priority. When the operation is invoked, the server will respond with the amount of available space on the queue. Examples of ‘elements’ are Octets, Audio Samples, digital waveform data words, etc.

4.2.1.1.1 Synopsis.

unsigned short spaceAvailable (in octet priorityQueueID);

4.2.1.1.2 Parameters.

priorityQueueID

This parameter indicates the PriorityQueue for which the Service Provider is required to return the amount of available queue space measured in ‘elements’. The number of priority queues is set up via SetNumOfPriorityQueues primitive. If SetNumOfPriorityQueues has not been called, the default number of priority queues is 1.

4.2.1.1.3 State.

Any state.

4.2.1.1.4 New State.

None.

4.2.1.1.5 Response.

spaceAvailable

This is the amount of available space, in ‘elements’, for the given priority queue ID. Examples of ‘elements’ are Octets, Audio Samples, digital waveform data words, etc.

4.2.1.1.6 Originator.

Service User.

4.2.1.1.7 Errors/Exceptions.

spaceAvailable

The Service Provider returns a value of –1 when the priority queue ID is not defined.

4.2.1.2 enableFlowControlSignals.

This operation is used to activate and deactivate the ‘water-mark’ signals. The default is false (signals will not be generated).

4.2.1.2.1 Synopsis.

oneway void enableFlowControlSignals (in boolean enable);

4.2.1.2.2 Parameters.

enable = TRUE

The Service Provider will signal the Lowwater and Highwater queue conditions, to the Service User, when the Lowwater mark has been reached (queue near empty).

enable =FALSE

The Service Provider will not generate signals to indicate the Lowwater and Highwater queue conditions. It is up to the Service User to poll the Service Provider to insure the Service Provider will not be starved or the queue will not overflow. The instantiating API should define behavior upon starvation or queue overflow.

4.2.1.2.3 State.

NO_PROVIDER_WATERMARK_SIGNALS or PROVIDER_WATERMARK_SIGNALS

4.2.1.2.4 New State.

enable = TRUE: PROVIDER_WATERMARK_SIGNALS

enable = FALSE: NO_PROVIDER_WATERMARK_SIGNALS

4.2.1.2.5 Response.

None.

4.2.1.2.6 Originator.

Service User.

4.2.1.2.7 Errors/Exceptions.

None.

4.2.1.3 enableEmptySignal.

This operation is used to activate and deactivate the 'empty' signal. The signal will not be generated when set to *enable* is False.

4.2.1.3.1 Synopsis.

oneway void enableEmptySignal (in boolean enable);

4.2.1.3.2 Parameters.

enable = TRUE

The Service Provider will generate a signal to the Service User when the all queues are empty.

enable = FALSE

The Service Provider will not generate a signal to indicate all queues are empty. It is up to the Service User to poll the Service Provider to insure the Service Provider's queue

will not be starved. The instantiating waveform API should define behavior upon starvation.

4.2.1.3.3 State.

NO_PROVIDER_EMPTY_SIGNAL or PROVIDER_ EMPTY_SIGNAL

4.2.1.3.4 New State.

enable = TRUE PROVIDER_EMPTY_SIGNAL

enable = FALSE NO_PROVIDER_EMPTY_SIGNAL

4.2.1.3.5 Response.

None.

4.2.1.3.6 Originator.

Service User.

4.2.1.3.7 Errors/Exceptions.

None.

4.2.1.4 setNumOfPriorityQueues.

This operation is used by the Service User to inform the Service Provider how many priority queues to provide.

4.2.1.4.1 Synopsis.

oneway void setNumOfPriorityQueues (in octet numOfPriorities);

4.2.1.4.2 Parameters.

numOfPriorites

Specifies the number of priority queues the Service Provider must provide. If the value is set to 10, the range of the pushPacket priority parameter is 0-9, with 9 being the highest priority. Messages with a priority of 9 will be processed first by the Service Provider.

4.2.1.4.3 State.

Any state.

4.2.1.4.4 New State.

Same state.

4.2.1.4.5 Response.

None.

4.2.1.4.6 Originator.

Service User.

4.2.1.4.7 Errors/Exceptions.

None.

4.2.1.5 getMaxPayLoadSize.

This attribute allows a Service User to interrogate the maximum payload size, in ‘elements’, provided by a Service Provider. Examples of ‘elements’ are Octets, Audio Samples, digital waveform data-words, etc.

4.2.1.5.1 Synopsis.

readonly attribute unsigned short maxPayloadSize;

maxPayLoadSize is an attribute, whose "Set" and "Get" functions are auto-generated.

4.2.1.5.2 Parameters.

None.

4.2.1.5.3 State.

Any state.

4.2.1.5.4 New State.

Same state.

4.2.1.5.5 Response.

maxPayloadSize

> Specifies the maximum number of traffic ‘elements’ allowed per pushPacket call. It is established by the Packet Service Provider. Examples of ‘elements’ are Octets, Audio Samples, digital waveform data-words, etc.

4.2.1.5.6 Originator.

Service User.

4.2.1.5.7 Errors/Exceptions.

None.

4.2.1.6 getMinPayLoadSize.

This attribute allows a Service User to interrogate the minimum payload size, in ‘elements’, provided by a Service Provider. Examples of ‘elements’ are Octets, Audio Samples, digital waveform data-words, etc.

4.2.1.6.1 Synopsis.

readonly attribute unsigned short minPayloadSize;

minPayLoadSize is an attribute, whose "Set" and "Get" functions are auto-generated.

4.2.1.6.2 Parameters.

None.

4.2.1.6.3 State.

Any state.

4.2.1.6.4 New State.

Same state.

4.2.1.6.5 Response.

minPayloadSize

Specifies the minimum number of traffic 'elements' allowed per pushPacket call. It is established by the Packet Service Provider. Examples of 'elements' are Octets, Audio Samples, digital waveform data-words, etc.

4.2.1.6.6 Originator.

Service User

4.2.1.6.7 Errors/Exceptions.

None.

4.2.1.7 pushPacket.

pushPacket provides the capability to push data packets from a Service User to a Service Provider or from a Service Provider to a Service User. A packet is made up of two parts, control and payload. The payload is queued according to the priority and is processed according to the information specified in control parameter.

4.2.1.7.1 Synopsis.

```
oneway void pushPacket (
                        in octet priority,
                        in AudioControlType control,
                        in CF::UlongSequence payload
                        );
```

4.2.1.7.2 Parameters.

priority:

conveys to the Service Provider the number of the priority queue where the packet is to be queued. (See setNumOfPriorityQueues)

control:

conveys the ControlType defined by the instantiating waveform API.

The control parameter has the following structures, which depend on the direction data is being pushed, downstream (towards the antenna) or upstream:

```
struct AudioControlType  {
        boolean endOfStream;
        unsigned short streamID;
        octet sequenceNum;
        };
```

endOfStream

Indicates the last symbol in this packet is the end of this stream sequence.

streamID

Associates a packet with a particular packet stream sequence. A stream sequence might include all packets received during a single reception or transmitted during a single transmission.

sequenceNum

Conveys the sequence number of a packet within a stream of packets having the same stream ID.

Indicates Start of Stream when the value of sequenceNum is zero (e.g., first packet in having this stream ID). The waveform application is responsible for setting this value to zero.

attribute1

This parameter is defined by the Waveform API.

payload:

This parameter is of type CF:: UlongSequence and it contains receive or transmit traffic data.

4.2.1.7.3 State.

Any state.

4.2.1.7.4 New State.

Same state.

4.2.1.7.5 Response.

None.

4.2.1.7.6 Originator.

Service User

4.2.1.7.7 Errors/Exceptions.

None.

4.2.2 Packet Signals Services.

These services provide flow control for the pushPacket operation.

4.2.2.1 signalHighWatermark.

This service is a call back event to the Service User indicating a queue has reached its high watermark. If priority or multiple queues are being supported, the priorityQueueID indicates which queue has reached its high watermark.

4.2.2.1.1 Synopsis.

oneway void signalHighWatermark (in octet priorityQueueID);

4.2.2.1.2 Parameters.

priorityQueueID

indicates the queue priority which has reached its high water mark. (See setNumOfPriorityQueues).

4.2.2.1.3 State.

Any state.

4.2.2.1.4 New State.

Same state.

4.2.2.1.5 Response.

None.

4.2.2.1.6 Originator.

Service Provider.

4.2.2.1.7 Errors/Exceptions.

None.

4.2.2.2 signalLowWatermark.

This service is a call back event to the Service User indicating a queue has reached its low watermark. If priority or multiple queues are being supported, the priorityQueueID indicates which queue has reached its low watermark.

4.2.2.2.1 Synopsis.

oneway void signalLowWaterMark (in octet priorityQueueID);

4.2.2.2.2 Parameters.

priorityQueueID

indicates the queue priority which has reached its low water mark. (See setNumOfPriorityQueues).

4.2.2.2.3 State.

Any state.

4.2.2.2.4 New State.

Same state.

4.2.2.2.5 Response.

None.

4.2.2.2.6 Originator.

Service Provider.

4.2.2.2.7 Errors/Exceptions.

None.

4.2.2.3 signalEmpty.

This service is a call back event to the Service User indicating the queue is empty. If priority or multiple queues are being supported, it indicates all queues are empty.

4.2.2.3.1 Synopsis.

oneway void signalEmpty ();

4.2.2.3.2 Parameters.

None.

4.2.2.3.3 State.

Any state.

4.2.2.3.4 New State.

Same state.

4.2.2.3.5 Response.

None.

4.2.2.3.6 Originator.

Service Provider.

4.2.2.3.7 Errors/Exceptions.

None.

4.2.3 signalError Service.

This service is a call back to the Service User indicating the Service Provider has detected an error in a pushPacket payload.

4.2.3.1.1 Synopsis.

void signalError(in IOError_T errorDetails);

4.2.3.1.2 Parameters.

None.

4.2.3.1.3 State.

Any operational state.

4.2.3.1.4 New State.

None.

4.2.3.1.5 Response.

errorDetails

conveys the following pushPacket payload error details to the Service User.

Error_InvalidData	indicates improperly formatted data
Error_BadData	indicates data exceeds the expected value
Error_CRC	indicates data failed a Cyclic Redundancy Code check

4.2.3.1.6 Originator.

I/O device.

4.2.3.1.7 Errors/Exceptions.

None.

5 ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES.

Attributes: Attributes may be set and fetched in any order. There is no order of precedence. The parameterized radio parameters should not be set unless all defined values have been initialized.

Tones and Beeps: Tones and beeps must be created before they can be started or sent. Starting an already started tone will have no effect on the audio device.

Sending an already active beep will extend the remaining beep duration by one complete beep duration.

Stopping already stopped tones will have no effect.

The sequence of operations used to control upstream flow of audio samples is defined Figure 3.4 Sequence Diagram, Voice.

6 PRECEDENCE OF SERVICE PRIMITIVES.

Where the precedence of service primitives are critical to proper operation, they are addressed in sections 3 and 4.

7 SERVICE USER GUIDELINES.

Auto-generated "get" and "set" operations are used to set and read attributes where they are defined in SCA Building Blocks. Otherwise, the use of attributes and their default "get"/"set" operations is avoided, as they do not support user-defined exceptions.

8 SERVICE PROVIDER-SPECIFIC INFORMATION.

Where Service Provider-specific information is critical to proper operation, it is addressed in sections 3 and 4.

9 IDL.

IDL is provide for the SINCGARS I/O API and the Audio API Building Block

9.1 I/O API.

The following IDL file describes the I/O API.

```
//Source file: c:/program files/devstudio/vc/atl/include/IO.idl

#ifndef __IO_DEFINED
#define __IO_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

/* This package provides the main framework for all objects within the radio.
*/

module CF {

      /* This type is a CORBA  IDL struct type which can be used to hold any
CORBA basic type or static IDL type.  */

      struct DataType {
            /* The id attribute indicates the kind of value and type (e.g.,
frequency, preset, etc.).  The id can be an UUID string, an integer string,
or a name identifier.  */
            string id;
            /* The value attribute can be any static IDL type or CORBA basic
type.  */
            any value;
      };

      /* The Properties is a CORBA IDL unbounded sequence of DataType(s),
which can be used in defining a sequence of name and value pairs.  The
relationships for Propertiies are shown in the Properties Relationships
figure.  */

      typedef sequence <DataType> Properties;

      /* This type is a CORBA unbounded sequence of octets.  */

      typedef sequence <octet> OctetSequence;

      /* This type defines a sequence of strings */

      typedef sequence <string> StringSequence;

      /* The LifeCycle interface defines the generic operations for
initializing or releasing an instantiated component specific data and/or
processing elements.  */

      interface LifeCycle {
```

```
            /* This exception indicates an error occurred during component
initialization.  The messages provides additional information describing the
reason why the error occurred.  */

            exception InitializeError {
                  StringSequence errorMessages;
            };

            /* This exception indicates an error occurred during component
releaseObject.  The messages provides additional information describing the
reason why the errors occurred.  */

            exception ReleaseError {
                  StringSequence errorMessages;
            };

            /* The purpose of the initialize operation is to provide a
mechanism to set an object to an known initial state.  (For example, data
structures may be set to initial values, memory may be allocated, hardware
components may be configured to some state, etc.).

The initialize operation raises an exception if initialized more than once
for a component.  Initialization behavior is implementation dependent.

This operation raises the InitializeError when an initialization error
occurs.
            @roseuid 39EB58810202 */
            void initialize ()
                  raises (InitializeError);

            /* The purpose of the releaseObject operation is to provide a
means by which an instantiated component may be torn down.  The releaseObject
operation releases itself from the CORBA ORB.

The releaseObject operation releases all internal memory allocated by the
component during the life of the component.

This operation raises a ReleaseError when a release error occurs.
            @roseuid 39EB58810203 */
            void releaseObject ()
                  raises (ReleaseError);

      };

      /* The TestableObject interface defines operations that can be used to
test object implementations.  */

      interface TestableObject {
            /* This exception indicates the test is unknown by the component.
*/

            exception UnknownTest {
            };

            /* The runTest operation allows components to be "blackbox"
tested.  This allows Built-In Test (BIT) to be implemented as well as
```

```
provides a mean to isolate faults (both software and hardware) within the
system.

The runTest operation uses the testNum argument to specify the
implementation-specific test to be run.  Tests to be implemented by a
component are component-dependent and are specified in the component's
software profile.

An UnknownTest exception shall be raised when no such test is known by the
component.
            @roseuid 39EB5881024A */
            long runTest (
                  in unsigned long testNum
                  )
                  raises (UnknownTest);

      };

      /* The PropertySet interface defines configure and query operations to
access component properties/attributes.  */

      interface PropertySet {
            /* This exception indicates the configuration of a component has
failed (no configuration at all was done).  The message provides additional
information describing the reason why the error occurred.  The invalid
properties returned indicates the properties that were invalid.  */

            exception InvalidConfiguration {
                  Properties invalidProperties;
                  string msg;
            };

            /* This exception indicates the configuration of a component was
partially successful.  The invalid properties returned indicates the
properties that were invalid.  */

            exception PartialConfiguration {
                  Properties invalidProperties;
            };

            /* This exception indicates a set of properties unknown by the
component.  */

            exception UnknownProperties {
                  Properties invalidProperties;
            };

            /* The purpose of this operation is to allow id/value pair
configuration properties to be assigned to components implementing this
interface. ^

The configure operation shall assign values to the properties as indicated in
the configProperties argument.  An component's SPD profile indicates the
valid configuration values.
```

```
This operation raises InvalidConfiguration exception when a configuration
error occurs preventing any property configuration on the component.

This operation raises PartialConfiguration exception when some configuration
properties were successful and some configuration properties were not
successful.
            @roseuid 39EB5881024F */
            void configure (
                  in Properties configProperties
                  )
                  raises (InvalidConfiguration, PartialConfiguration);

            /* The purpose of this operation is to allow a component to be
queried to retrieve its properties.

If the configProperties are zero size then, the query operation returns all
component properties.  If the configProperties are not zero size, then the
query operation returns only those id/value pairs specified in the
configProperties.  An component's SPD profile indicates the valid query
types.

This operation raises the UnknownProperties exception when one or more
properties being requested are not known by the component.
            @roseuid 39EB58810251 */
            void query (
                  inout Properties configProperties
                  )
                  raises (UnknownProperties);

      };

      /* The Resource interface provides a common API for the control and
configuration of a software component.
      The Resource interface inherits from the LifeCycle, PropertySet, and
TestableObject interfaces.
      The inherited LifeCycle, PropertySet, and TestableObject interface
operations are documented in their respective sections of this document.
      The CF Resource interface may also be inherited by other application
interfaces as described in the Software Profile's Software Component
Descriptor (SCD) file.  */

      interface Resource : LifeCycle, TestableObject, PropertySet {
            /* This exception is raised if an undefined port is requested.
*/

            exception UnknownPort {
            };

            /* This exception indicates a Start error has occurred for the
Resource.  An error message is given explainng the start error.  */

            exception StartError {
                  string msg;
            };
```

```
            /* This exception indicates a Stop error has occurred for the
Resource.  An error message is given explainng the stop error.  */

            exception StopError {
                  string msg;
            };

            /* The start operation enables operations for the Resource.
            @roseuid 39EB58810163 */
            void start ()
                  raises (StartError);

            /* The start operation disables operations for the Resource.
            @roseuid 39EB58810164 */
            void stop ()
                  raises (StopError);

            /* The getPort operation provides a mechanism to obtain a
specific consumer or producer Port.  A Resource may contain zero to many
consumer and producer port components.  The exact number is specified in the
component's Software Profile SPD's SCD.  These Ports can be either push or
pull types.  Multiple input and/or output ports provide flexibility for
Applications and Resources that must manage varying priority levels and
categories of incoming and outgoing messages, provide multi-threaded message
handling, or other special message processing.

The getPort operations returns the object reference to the named port as
stated in the Resource's SCD.

The getPort operation raises an UnknownPort exception if the port name is
invalid.
            @roseuid 39EB58810165 */
            Object getPort (
                  in string name
                  )
                  raises (UnknownPort);

      };

};

module IO_BB {

      typedef unsigned short IdType;

      struct BeepType {
            unsigned short durationInMs;
            unsigned short frequencyInHz;
            short beepLevelIndB;
      };

      interface IOSignals {
            /*
            @roseuid 3A1A85D0024D */
            void signalRTS (
                  in boolean RTS
```

```
                  );

      };

};

module PortTypes {

      /* This type is a CORBA unbounded sequence of unsigned longs.  */

      typedef sequence <unsigned long> UlongSequence;

      /* This type is a CORBA unbounded sequence of unsigned shorts.  */

      typedef sequence <unsigned short> UshortSequence;

};

module PacketBB {

      interface PacketSignals {
            /* This operation is a call event back to the PacketAPI client
indicating that a queue has reach the high watermark.  If priority or
multiple queues are being supported then the priorityQueueID indicates which
queue has reached the high watermark.
            @roseuid 39EF41B3009D */
            oneway void signalHighWatermark (
                  in octet priorityQueueID
                  );

            /* This operation is a call event back to the PacketAPI client
indicating that the queue has reach the low watermark.   If priority or
multiple queues are being supported then this indicates that the sum total of
all the queues has reached the low watermark.
            @roseuid 39EF41B3009F */
            oneway void signalLowWaterMark (
                  in octet priorityQueueID
                  );

            /* This operation is a call event back to the PacketAPI client
indicating that the queue has emptied.   If priority or multiple queues are
being supported then this indicates that the sum total of all the queues has
reached zero.
            @roseuid 39EF41B300A1 */
            oneway void signalEmpty ();

      };

};

module IO {

      module Audio {

            struct CVSDParameterType {
                  unsigned short DataSamplesize;
```

```
            boolean enable;
      };

      struct AudioConfigurationType {
            CVSDParameterType CVSDParameter;
            string Version;
            string AudioModuleName;
            unsigned short AudioChannelNumber;
            boolean LPC10Enable;
      };

      /* Used to configure the IO device */

      interface AudioIOConfiguration {
            attribute AudioConfigurationType configuration;
      };

      struct AudioIOControlType {
            unsigned short SideToneGainIndB;
            unsigned short AGCAttackTimeInmS;
            unsigned short AGCReleaseTimeInmS;
            unsigned short AGCDynamicRangeIndB;
            boolean AudioOutputEnabled;
            boolean SidetoneEnabled;
            short OutputGainIndB;
            short MicrophoneGainIndB;
      };

      enum TrafficType {
            SCPT,             /* Gen_Plain_Text_Mode_Msg #32 */
            FHPT,
            FHCT,
            SCCT              /* Gen_Cipher_ Text_Mode_Msg #31 */
      };

      /* Controls the operational parameters associated with a specific
I/O device.  */

      interface AudioIOControl {
            attribute AudioIOControlType parameters;
            attribute boolean txActive;
            attribute boolean rxActive;

            /*
            @roseuid 3A1A8BB20360 */
            void enableRTS_CTS (
                  in boolean Enable
                  );

            /*
            @roseuid 3A1A8BB2036B */
            void setCTS (
                  in boolean CTS
                  );

            /*
```

```
                @roseuid 3A240E4B0179 */
                void txBegin (
                      in TrafficType TxTraffic
                      );

                /*
                @roseuid 3A240E7D01FD */
                void rxBegin (
                      in TrafficType RxTraffic
                      );

          };

          enum FlowConfigurationType {
                HWFlowControl,
                XonXoffControl,
                None
          };

          struct SyncDataType {
                unsigned short dataRateInHz;
          };

          struct AsyncDataType {
                unsigned short dataRateInHz;
                unsigned short NumberOfStopBits;
                boolean ParityBit;
                unsigned short NumberOfDataBits;
                unsigned short NumberOfStartBits;
          };

          struct DataIOControlType {
                boolean DDMCActive;
                boolean ADMCActive;
          };

          /* The IO control Service Group defines the interface for
controlling the operational parameters associated with Audio and Data
Devices.  */

          interface DataIOControl {
                attribute DataIOControlType parameters;
                attribute boolean txActive;
                attribute boolean rxActive;

                /*
                @roseuid 3A242182015E */
                void enableRTS_CTS (
                      in boolean Enable
                      );

                /*
                @roseuid 3A2421820172 */
                void setCTS (
                      in boolean CTS
                      );
```

```
      /*
      @roseuid 3A269D1E0190 */
      void txBegin (
            in TrafficType TxTraffic
            );

      /*
      @roseuid 3A269D2402B1 */
      void rxBegin (
            in TrafficType RxTraffic
            );

};

struct AlarmType {
      unsigned short FrequencyInHz;
      unsigned short OnTimeInmS;
      unsigned short OffTimeInmS;
      short ToneLevelIndB;
};

struct MultiToneAlarmType {
      unsigned short numberOfTones;
      AlarmType Tone1;
      AlarmType Tone2;
      AlarmType Tone3;
      unsigned short SeqRepeateTimeInmS;
};

enum IOErrorType {
      ErrorInvalidData,
      ErrorBadData,
      ErrorCRC
};

interface IOErrorSignal {
      /*
      @roseuid 3A2506CA032A */
      void signalError (
            in IOErrorType errorDetails
            );

};

struct AudioControlType {
      boolean endOfStream;
      unsigned short streamID;
      octet sequenceNum;
};

/* Mac octet  Packet */

interface OctetPacket {
      attribute unsigned short minPayloadSize;
```

```
                  /* The maxPacketSize is a read only attribute set by the
Packet Server and the get operation reports back the maximum number of
traffic units allowed in one pushPacket call.  */

                  attribute unsigned short maxPayloadSize;

                  /* This operation is used to push Client data to the Server
with a Control element and a Payload element.
                  @roseuid 39F998E602BA */
                  void pushPacket (
                        in octet priority,
                        in AudioControlType control,
                        in CF::OctetSequence payload
                        );

                  /* The operation returns the space available in the Servers
queue(s) in terms of the implementers defined Traffic Units.
                  @roseuid 39F998E602E3 */
                  short spaceAvailable (
                        in octet priorityQueueID
                        );

                  /* This operation allows the client to turn the High
Watermark Signal ON and OFF.
                  @roseuid 39F998E602EC */
                  void enableFlowControlSignals (
                        in boolean enable
                        );

                  /* This operation allows the client to turn theEmpty Signal
ON and OFF.
                  @roseuid 39F998E602EE */
                  void enableEmptySignal (
                        in boolean enable
                        );

                  /*
                  @roseuid 39F998E602F7 */
                  void setNumOfPriorityQueues (
                        in octet numOfPriorities
                        );

            };

            /* MAC unsigned long packet */

            interface UlongPacket {
                  attribute unsigned short minPayloadSize;
                  /* The maxPacketSize is a read only attribute set by the
Packet Server and the get operation reports back the maximum number of
traffic units allowed in one pushPacket call.  */

                  attribute unsigned short maxPayloadSize;

                  /* This operation is used to push Client data to the Server
with a Control element and a Payload element.
```

```
                  @roseuid 39F99B57021A */
                  void pushPacket (
                        in octet priority,
                        in AudioControlType control,
                        in PortTypes::UlongSequence payload
                        );

                  /* The operation returns the space available in the Servers
queue(s) in terms of the implementers defined Traffic Units.
                  @roseuid 39F99B57022D */
                  short spaceAvailable (
                        in octet priorityQueueID
                        );

                  /* This operation allows the client to turn the High
Watermark Signal ON and OFF.
                  @roseuid 39F99B570237 */
                  void enableFlowControlSignals (
                        in boolean enable
                        );

                  /* This operation allows the client to turn theEmpty Signal
ON and OFF.
                  @roseuid 39F99B570241 */
                  void enableEmptySignal (
                        in boolean enable
                        );

                  /*
                  @roseuid 39F99B570243 */
                  void setNumOfPriorityQueues (
                        in octet numOfPriorities
                        );

            };

            /* MAC unsigned short Packet */

            interface UshortPacket {
                  attribute unsigned short minPayloadSize;
                  /* The maxPacketSize is a read only attribute set by the
Packet Server and the get operation reports back the maximum number of
traffic units allowed in one pushPacket call.  */

                  attribute unsigned short maxPayloadSize;

                  /* This operation is used to push Client data to the Server
with a Control element and a Payload element.
                  @roseuid 39F99C500357 */
                  void pushPacket (
                        in octet priority,
                        in AudioControlType control,
                        in PortTypes::UshortSequence payload
                        );
```

```
                  /* The operation returns the space available in the Servers
queue(s) in terms of the implementers defined Traffic Units.
                  @roseuid 39F99C500363 */
                  short spaceAvailable (
                        in octet priorityQueueID
                        );

                  /* This operation allows the client to turn the High
Watermark Signal ON and OFF.
                  @roseuid 39F99C50036C */
                  void enableFlowControlSignals (
                        in boolean enable
                        );

                  /* This operation allows the client to turn theEmpty Signal
ON and OFF.
                  @roseuid 39F99C500375 */
                  void enableEmptySignal (
                        in boolean enable
                        );

                  /*
                  @roseuid 39F99C500377 */
                  void setNumOfPriorityQueues (
                        in octet numOfPriorities
                        );

            };

            interface User : UlongPacket, PacketBB::PacketSignals {
            };

            interface Provider : UlongPacket, PacketBB::PacketSignals,
IOErrorSignal {
            };

            enum DataDiscriminator {
                  SynchronousData,
                  AsynchronousData
            };

            union DataConfigurationType switch(DataDiscriminator) {
                  case SynchronousData: SyncDataType syncDataPort;
                  case AsynchronousData: AsyncDataType asyncDataPort;
            };

            /* Data IO Configuration */

            interface DataIOConfiguration {
                  attribute DataConfigurationType configuration;

                  /*
                  @roseuid 3A241CFD002E */
                  void setFlowConfig (
                        in FlowConfigurationType flowControl
                        );
```

```
            };

            enum AlarmDiscriminator {
                  CT,
                  PTOverride,
                  DataInHoldingMemory,
                  PacketBusy,
                  PTMode,
                  COMSECParity
            };

            union ToneProfileType switch(AlarmDiscriminator) {
                  case CT: AlarmType CTAlarm;
                  case PTOverride: AlarmType PTOverrideAlarm;
                  case DataInHoldingMemory: AlarmType
DataInHoldingMemoryAlarm;
                  case PacketBusy: AlarmType PacketBusyAlarm;
                  case PTMode: MultiToneAlarmType PTModeAlarm;
                  case COMSECParity: AlarmType COMSECParityAlarm;
            };

            enum BeepDiscriminator {
                  CTTraffic,
                  COMSECLoad,
                  FillZeroizedGood,
                  FHLoadTransfer,
                  Cue,
                  ZeroizeBad,
                  HoldingMemoryTransfer,
                  TransmitERF,
                  NetIDChange,
                  BitFail,
                  ActiveFail
            };

            union BeepProfileType switch(BeepDiscriminator) {
                  case CTTraffic: IO_BB::BeepType CTTrafficBeep;
                  case COMSECLoad: IO_BB::BeepType COMSECLoadBeep;
                  case FillZeroizedGood: IO_BB::BeepType
FillZeroizedGoodBeep;
                  case FHLoadTransfer: IO_BB::BeepType FHLoadTransferBeep;
                  case Cue: IO_BB::BeepType CueBeep;
                  case ZeroizeBad: IO_BB::BeepType ZeroizeBadBeep;
                  case HoldingMemoryTransfer: IO_BB::BeepType
HoldingMemoryTransferBeep;
                  case TransmitERF: IO_BB::BeepType TransmitERFBeep;
                  case NetIDChange: IO_BB::BeepType NetIDChangeBeep;
                  case BitFail: IO_BB::BeepType BitFailBeep;
                  case ActiveFail: IO_BB::BeepType ActiveFailBeep;
            };

            /* Provides the tone and beep sevice to the operator */

            interface Audible_AlertsAndAlarms {
                  attribute boolean txActive;
```

```
                  attribute boolean rxActive;

                  /*
                  @roseuid 3A1A8C6401F4 */
                  IO_BB::IdType createTone (
                        in ToneProfileType ToneProfile
                        );

                  /*
                  @roseuid 3A1A8C6401FE */
                  void startTone (
                        in IO_BB::IdType ToneId
                        );

                  /*
                  @roseuid 3A1A8C640209 */
                  void stopTone (
                        in IO_BB::IdType ToneId
                        );

                  /*
                  @roseuid 3A1A8C640213 */
                  void stopAllTones ();

                  /*
                  @roseuid 3A1A8C640230 */
                  IO_BB::IdType createBeep (
                        in BeepProfileType Beep
                        );

                  /*
                  @roseuid 3A1A8C640244 */
                  void sendBeep (
                        in IO_BB::IdType BeepId
                        );

            };

            interface Controller : CF::Resource, IO_BB::IOSignals,
Audible_AlertsAndAlarms, AudioIOConfiguration, AudioIOControl {
            };

      };

};

#endif
```

9.2 API BUILDING BLOCK.

The following IDL file describes the API building blocks.

```
//Source file: H:/JTRS/SYSJTRS/api/rose models/ITTBBIDL/APIBB.idl

#ifndef __APIBB_DEFINED
#define __APIBB_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

module APIBB {

        interface ChannelErrorControl {
                /*
                @roseuid 39EB177F0102 */
                void channelErrorControl (
                        in boolean ErrorControl
                        );

        };

        interface DropCapture {
                /*
                @roseuid 39C643F203D4 */
                boolean dropCapture ();

        };

        interface ReceiveTermination {
                /*
                @roseuid 39D0073C0171 */
                boolean dropCapture ();

                /*
                @roseuid 39D0ADCC0174 */
                boolean abortReceive ();

        };

        interface PhysicalManagement {
                readonly attribute unsigned short maxTU;
                readonly attribute unsigned short minTU;
        };

        interface TransmitInhibit {
                /*
                @roseuid 39D0B62B0021 */
                boolean inhibitTransmit (
                        in boolean Inhibit
```

```
                );

};

interface PacketSignals {
        /* This operation is a call event back to the PacketAPI client indicating that a queue has
        reach the high watermark.  If priority or multiple queues are being supported then the
        priorityQueueID indicates which queue has reached the high watermark.
        @roseuid 38F3442F01B8 */
        oneway void signalHighWatermark (
                in octet priorityQueueID
                );

        /* This operation is a call event back to the PacketAPI client indicating that the queue has
        reach the low watermark.  If priority or multiple queues are being supported then this
        indicates that the sum total of all the queues has reached the low watermark.
        @roseuid 38F3446F025A */
        oneway void signalLowWaterMark (
                in octet priorityQueueID
                );

        /* This operation is a call event back to the PacketAPI client indicating that the queue has
        emptied.  If priority or multiple queues are being supported then this indicates that the
        sum total of all the queues has reached zero.
        @roseuid 38FE26CF02FA */
        oneway void signalEmpty ();

};

interface IOSignals {
        /*
        @roseuid 39E4A192016E */
        oneway void SignalRTS (
                in boolean RTS
                );

};

struct BeepType {
        unsigned short frequencyInHz;
        unsigned short durationInMs;
        short BeepLevelIndB;
};

enum ErrType {
        PktUsageErr,
        PktErrNo
};

typedef unsigned short IdType;
```

```
        /* Identify this data stream for acknowledgement processing, cancelation of transmission,etc.  */

        struct StreamControlType {
                /* Indicates that the last symbol of this hop is an end of stream.  */
                boolean endOfStream;
                unsigned short streamID;
                /* Sequence number of the hop within the stream sequence.  The waveform application
                sets this value to zero at every occurrence of a start of stream.  If value is set to zero it
                indicates beginning of stream.  */
                octet sequenceNum;
        };

        struct TimeType {
                unsigned long seconds;
                unsigned long nanoSec;
        };

};

#endif
```

10 UML.

UML class and component diagrams are shown in the following figures.

10.1 CONTROLLER DIAGRAM.

Figure 10-1. UML Controller Relationships

10.2 USER DIAGRAM.

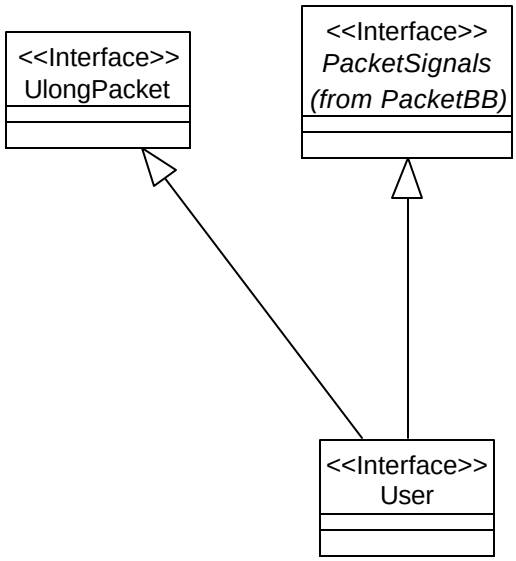

Figure 10-2. UML User, Data Packet Relationships

10.3 PROVIDER DIAGRAM.

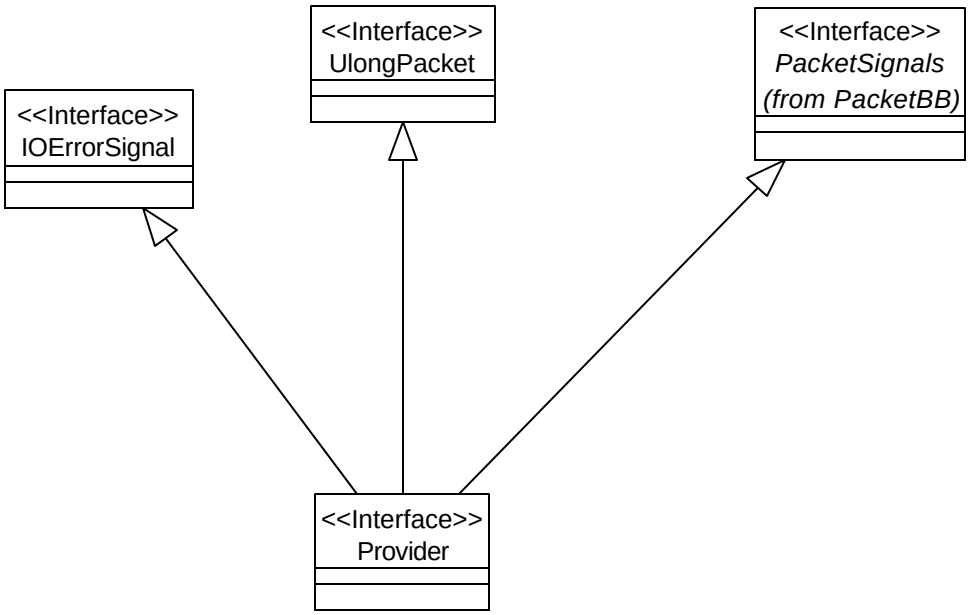

Figure 10-3. UML Provider, Data Packet Relationships

10.4 ULONGPACKET DIAGRAM.

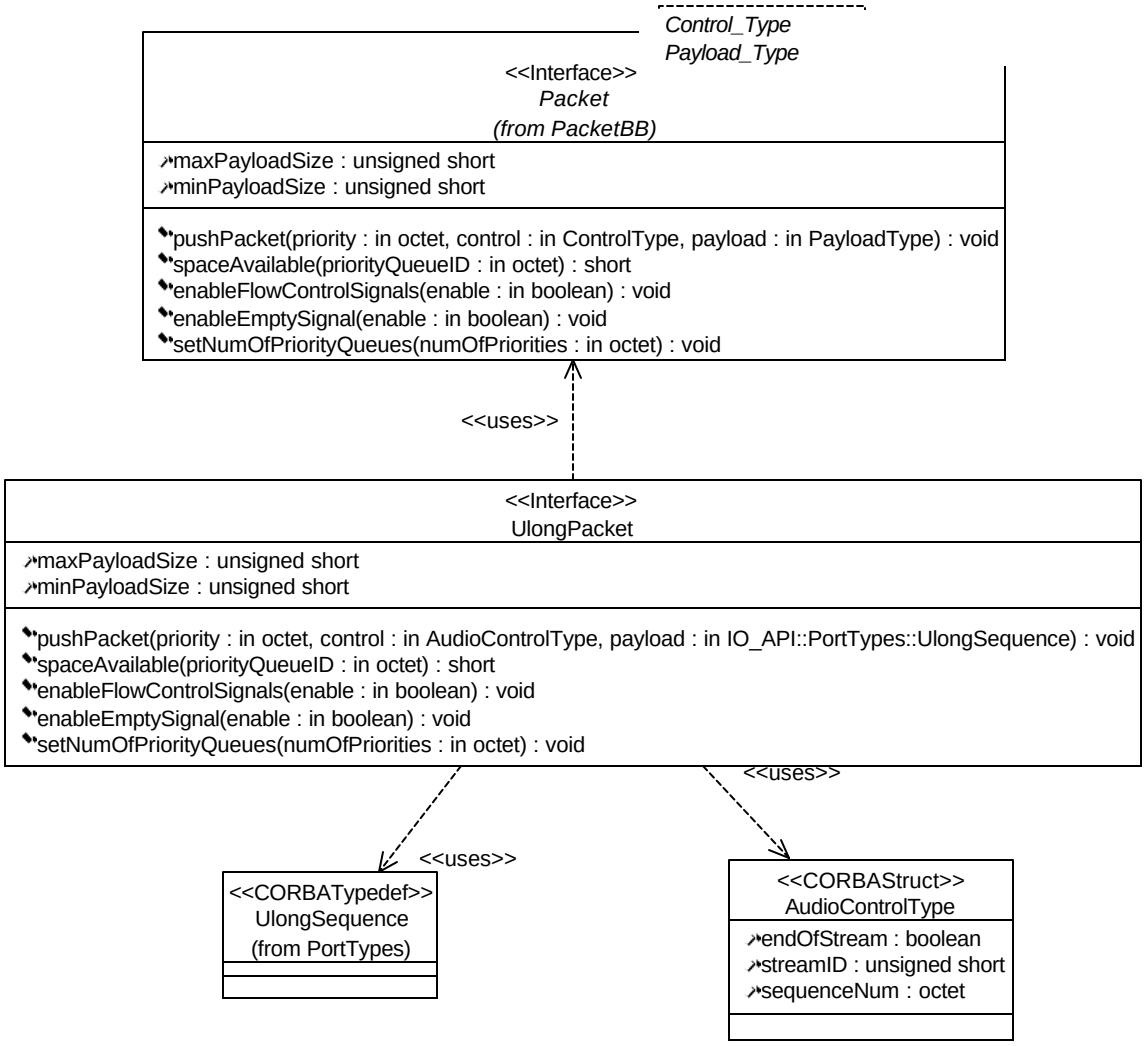

Figure 10-4. UML UlongPacket Relationships

10.5 COMPONENT DIAGRAM.

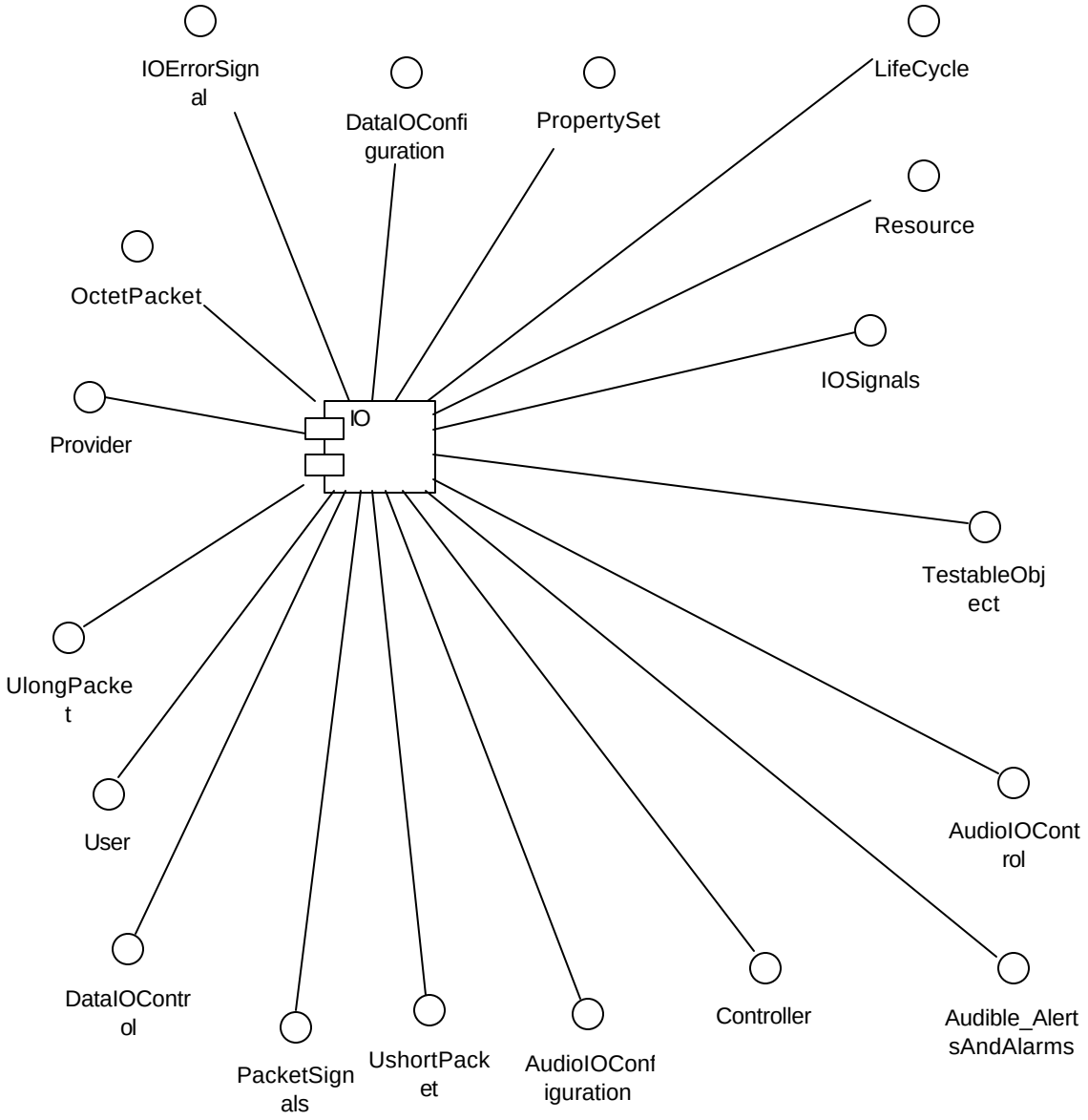

Figure 10-5. Component Diagram